UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Arquitetura Autonômica para a Alocação de Recursos Através de Migração de Serviços em Ambientes Fog Computing

Dissertação de Mestrado

Danilo Souza Silva

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Danilo Souza Silva

Uma Arquitetura Autonômica para a Alocação de Recursos Através de Migração de Serviços em Ambientes Fog Computing

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Admilson de Ribamar Lima Ribeiro
Coorientador(a): Edward David Moreno Ordonez

São Cristóvão – Sergipe

2019

Danilo Souza Silva
Uma Arquitetura Autonômica para a Alocação de Recursos Através de Migração de Serviços em Ambientes Fog Computing/ Danilo Souza Silva. – São Cristóvão – Sergipe, 2019-
115 p. : il. (algumas color.) ; 30 cm.

Orientador: Admilson de Ribamar Lima Ribeiro
Coorientador: Edward David Moreno Ordonez

Dissertação de Mestrado – UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, 2019.

1. Fog Computing. 2. Computação Autonômica. 3. Sistemas Distribuídos. 4. Provisionamento de Recursos. 5. IoT. I. Orientador. II. Universidade Federal de Sergipe. III. Centro de Ciências Exatas e Tecnologia IV. Uma Arquitetura Autonômica para a Alocação de Recursos em ambientes Fog Computing

CDU 02:141:005.7

“Dedico este trabalho primeiramente a Deus, por ser essencial em minha vida, autor de meu destino, meu guia, socorro presente nas horas difíceis. Dedico também a minha esposa Rejane, às minhas filhas Joana e Eva, aos meus pais Ivan e Maria e aos meus irmãos Ivo e Tiago.”

Agradecimentos

Meus sinceros agradecimentos ao meu professores, Admilson Ribeiro e Edward David Moreno, por sua amizade e conhecimento a mim proporcionados. Pelo encorajamento, apoio, orientação e paciência desde o início dos meus primeiros passos nesta jornada. Pessoas das quais levarei os ensinamentos e as lembranças comigo pelo resto da vida.

A minha esposa, Rejane, por sempre fornecer constante incentivo nas minhas realizações e por, acima de tudo, partilhar do seu amor e amizade durante todos esses anos juntos, cheios de desafios e conquistas.

Aos meus pais Ivan Batista e Maria Souza, pelo seu amor incondicional e por tudo que fizeram por mim para me tornar capaz de buscar as minhas próprias conquistas. Aos meus irmãos Ivo e Tiago Souza por existirem em minha vida.

Estendo a minha gratidão aos amigos e colegas Marconi Oliveira, José Machado, Reneilson Carvalho, Walter do Espírito Santo e Adauto Cavalcante pelas contribuições de grande valor, apoio técnico e companheirismo durante o período do mestrado.

Aos professores e funcionários do DCOMP/UFS e também a todos que contribuíram diretamente e indiretamente para que eu chegasse até aqui.

Por fim, ao Programa de Pós-Graduação em Ciência da Computação e a Universidade Federal de Sergipe, pela oportunidade a mim concedida de buscar e contribuir com o conhecimento.

“A persistência é o menor caminho do êxito”. (Charles Chaplin)

Resumo

Nos últimos anos, o número de dispositivos inteligentes tais como, *smartphones*, sensores e veículos autônomos, vem crescendo de forma significativa. Nesse cenário, a demanda computacional necessária para atender a demanda de aplicações sensíveis à latência em domínios como IoT, Industria 4.0 e *smart cities* também está crescendo e o tradicional modelo de computação em nuvem já não é capaz de atender sozinho a todas as necessidades deste tipo de aplicação. Como alternativa para esta limitação, foi introduzido um novo paradigma de computação chamado *fog computing*. Este paradigma define a arquitetura que estende a capacidade computacional e o armazenamento da nuvem para a borda da rede. Contudo, um dos principais problemas é como determinar de forma eficiente, onde os serviços serão alocados de modo a atender determinadas necessidades de QoS para o provimento de serviços através de aplicações IoT. O presente estudo, tem como objetivo apresentar uma estratégia de otimização para o problema de alocação de recursos utilizando migração de serviços através de um modelo de arquitetura autonômico, baseado no laço de controle MAPE-K. A partir do modelo apresentado, a estratégia foi implementada com tecnologia de virtualização em contêineres e avaliada em um ambiente virtual de larga escala para IoT, denominado VIoLET. Os resultados mostram que é possível otimizar um ambiente de *fog computing* utilizando a migração de serviços entre os nós de acordo com objetivos estabelecidos e de forma autonômica. O trabalho contribui com uma revisão bibliográfica do estado da arte sobre gerenciamento de recursos, a implementação de um ambiente de monitoramento e orquestração para o VIoLET, além de contribuir com o desenvolvimento e avaliação da estratégia de otimização bem como, a análise da utilização de recursos da solução proposta. Por fim concluímos o trabalho apresentando uma lista de promissoras direções de pesquisas em linhas gerais para trabalhos futuros.

Palavras-chave: Fog Computing, Computação Autonômica, Sistemas Distribuídos, Provisionamento de Recursos, IoT.

Abstract

In recent years, the number of smart devices (eg smartphones, sensors, autonomous vehicles) has grown significantly. In this scenario, the computational demand needed to meet the demand for latency-sensitive applications in domains such as IoT, Industria 4.0 and smart cities is also growing and the traditional cloud computing model is no longer able to meet all of its needs alone of application. As an alternative to this limitation, a new computing paradigm called Fog Computing was introduced. This paradigm defines the architecture that extends the computational capacity and storage of the cloud to the edge of the network. However, one of the main problems is how to efficiently determine where services will be allocated to meet certain QoS requirements for provisioning services through IoT applications. The present study aims to present an optimization strategy for the resource allocation problem using service migration through an autonomic architecture model based on the MAPE-K control loop. Based on the presented model, the strategy was implemented with container virtualization technology and evaluated in a large scale virtual environment for IoT, called VIoLET. The results show that it is possible to optimize a fog computing environment using the service migration between the nodes according to established objectives and autonomously. The work contributes with a bibliographical review of the state of the art on resource management, the implementation of a monitoring and orchestration environment for VIoLET, as well as contributing to the development and evaluation of the optimization strategy as well as the analysis of resource utilization of the proposed solution. Finally, we conclude the paper by presenting a list of promising research directions outlined for future work. We expect the work to serve as a basis for research that seeks to develop optimization techniques for resource utilization in Fog Computing environments.

Keywords: Fog Computing, Autonomic Computing, Distributed Systems, Resource Provisioning, IoT.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet of Things
DC	Data Center
QoS	Quality of Service
WSN	Wireless Sensor Network
IaaS	Infrastructure as a Service
Paas	Plataform as a Service
SaaS	Software as a Service
API	Application Programming Interface
GA	Genetic Algorithm
CoAP	Constrained Application Protocol
MQTT	Message Queuing Telemetry Transport
SDN	Software Defined Networking
IEEE	Institute of Electrical and Electronics Engineers
TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
VM	Virtual Machine

Sumário

1 Introdução

1.1 Motivação

A Internet das Coisas (IoT) é um paradigma onde dispositivos inteligentes (ex. *smartphones*, *tablets*, sensores e veículos autônomos), são capazes de comunicar-se uns com os outros cooperando entre si para alcançar objetivos comuns (ATZORI; IERA; MORABITO, 2010). Estima-se que em meados de 2020 existirão mais de 24 bilhões de dispositivos inteligentes conectados, causando um impacto econômico de trilhões de dólares (GUBBI et al., 2013; RIVERA; MEULEN, 2014). De acordo com Loghin et al. (2015), o aumento do volume, variedade e velocidade do dados gerados por estes dispositivos requer um maior dimensionamento de recursos de data Center (DC), pois as características intrínsecas a este paradigma trazem diversos desafios para o desenvolvimento de aplicações que buscam acessar, integrar e analisar a quantidade de dados produzida por estes dispositivos.

Nesta direção, a computação em nuvem surgiu como uma alternativa para resolver estes problemas, oferecendo armazenamento sob demanda e escalável, bem como serviços de processamento que podem se adaptar aos requisitos da IoT. Contudo, para processar o grande volume de informações na infraestrutura tradicional da computação em nuvem, pode ocorrer um longo tempo de resposta e um maior consumo de largura de banda (BONOMI et al., 2014). Em aplicações sensíveis a latência, tais como, monitoramento de saúde (SUNG; CHIANG, 2012), veículos autônomos (GERLA et al., 2014) e sistemas de emergência (XU et al., 2014), o atraso causado pela transferência de dados ou a indisponibilidade da comunicação com o ambiente de nuvem podem ser inaceitáveis (BONOMI et al., 2014; DASTJERDI; BUYYA, 2016). Além disso, a segurança em um ambiente IoT também necessita de respostas rápidas uma vez que, em determinados tipos de ataque, pode não haver tempo suficiente para aguardar a resposta a determinados tipos ataque utilizando o processamento em um data center da computação em nuvem (MELLO et al., 2016).

Para superar essas limitações, o paradigma *fog computing* ou computação em névoa (COMPUTING, 2016) foi proposto pela Cisco em 2012 (MACHADO; MORENO; RIBEIRO, 2017). Este conceito amplia a noção de computação na nuvem e os requisitos de processamento, rede e armazenamento podem ser atendidos na borda da rede. Devido a estas características, a *fog computing* emergiu como uma solução promissora para permitir o processamento contínuo de dados na proximidade do usuário em tempo real. Contudo, trabalhos de pesquisa relacionados a *fog computing* encontram-se em seu estágio inicial de desenvolvimento e diversos desafios precisam ser superados (TOCZÉ; NADJM-TEHRANI, 2018).

1.2 Problemática e Hipótese

De acordo com Gupta et al. (2016), um dos principais problemas da *fog Computing* é o de desenvolver sistemas que consigam determinar de forma eficiente quais módulos de aplicação serão apresentados na borda da rede para cada dispositivo de acordo com suas restrições de recursos e necessidades de QoS. Além disso, em contextos muito ativos e dinâmicos como o da IoT, a competição por usar serviços compartilhados, e os recursos computacionais subjacentes alocados nos *gateways* (ex: memória, processador, armazenamento), podem levar o ambiente a eventos imprevisíveis, como indisponibilidade do serviço, tempo de resposta elevado e diminuição da confiabilidade (MAHMUD; KOTAGIRI; BUYYA, 2018).

Nesta direção, a utilização eficiente de recursos tornou-se uma das principais necessidades que devem ser enfrentadas pelos operadores de data centers e provedores de serviços em nuvem (SAHNI; VARMA, 2012). Esse problema foi resolvido com o advento da migração ao vivo. A migração ao vivo é suportada por plataformas de virtualização tradicionais, plataformas baseada em *hypervisor* e plataformas baseada em contêineres. Este conceito ajuda a melhorar a disponibilidade, a confiabilidade, a capacidade de transporte e a capacidade de gerenciamento de aplicações localizados em hardware que não são confiáveis (ROMERO; HACKER, 2011). Por outro lado, como alternativa para do gerenciamento eficiente de recursos, a IBM publicou em (COMPUTING et al., 2006) um modelo para desenvolvimento de sistemas autônomos, conhecido como MAPE-K. Este modelo engloba as fases de Monitoramento, Análise, Planejamento e Execução. Tais etapas compartilham uma base de conhecimento na qual é fundamental para a tomada de decisões.

Neste contexto, para nortear o desenvolvimento desta pesquisa formulou-se a seguinte questão de propósito geral:

Q1: Como os serviços podem ser alocados através dos nós de um ambiente *fog computing* de modo a proporcionar a otimização da utilização de recursos?

Como hipótese temos que a otimização da utilização de recursos pode ser alcançada através da adoção de estratégias eficientes para alocação de recursos através da migração dos serviços, utilizando como base, informações relevantes do ambiente orquestrado para a tomada

de decisões.

1.3 Justificativa

No âmbito desta área, este trabalho teve como ponto de partida, a análise do *Gartner Hype Cycle* (WALKER; BURTON; CANTARA, 2016). O Hype *Cycle* é a forma gráfica de representar a maturidade e a adoção de determinadas tecnologias (PRINSLOO; DEVENTER, 2017). O *Hype Cycle* é composto por cinco diferentes fases e cada uma das fases mostra os tópicos de tendências atuais de pesquisa e seu estágio atual. A primeira fase começa com um gatilho de inovação, seguido por um pico de expectativas infladas, em seguida o vale de desilusão, um esclarecimento de escopo e termina com um patamar de produtividade. De acordo com a *Gartner Inc.* a maioria das inovações é forçada a passar por essas fases durante sua vida útil.

O relatório anual de *Hype Cycle* (WALKER; BURTON, 2017), adicionalmente, apresenta uma análise de tendências apontando as principais tendências para tecnologias emergentes. Nessa análise, os campos de tecnologias de nuvem e de informações móveis começam a sair da fase de desilusão, enquanto tecnologias como IoT, computação de borda e veículos autônomos estão localizados em torno do pico das expectativas infladas. Consequentemente, a computação em nuvem e as tecnologias de informação móvel já foram bem pesquisadas e ganharam importância no mercado. Por outro lado, tecnologias como a IoT e computação de borda estão enfrentando uma fase crucial com muitos desafios ligados aos esforços de pesquisa (GUBBI et al., 2013; SHI et al., 2016).

Tecnologias como computação de borda e abordagens relacionadas, por exemplo, computação em nuvem móvel e computação de borda móvel, estão situadas próximas das plataformas IoT no gráfico. A declaração resultante desta análise de megatendências é que essas tecnologias estão atualmente no auge das expectativas, onde uma avaliação realista da extensão completa dos tópicos é difícil. Assim, este pico é seguido por uma descoberta de desafios conectados e problemas atuais a serem abordados até que as primeiras soluções aplicáveis sejam pesquisadas com sucesso (SHI et al., 2016).

Devido à proliferação de dispositivos de IoT geradores de dados, tais como, sensores e *smartphones*, a enorme quantidade de dados enviados pela rede é um desafio crescente para futuras aplicações (LOGHIN et al., 2015). Não apenas o processamento de dados é necessário, mas também a transmissão de dados dos dispositivos para os *data centers*.

Diante da necessidade de transmitir e processar a abundância futura de dados recebidos por dispositivos amplamente distribuídos, faz-se necessário um novo paradigma de computação. Este paradigma em evolução é a *fog computing*. Em contraste, os data centers utilizados na computação em nuvem não são tão distribuídos frequentemente encontram-se em locais distantes do usuário, resultando em tempos de resposta relativamente mais altos. É importante mencionar, que a *fog computing* não é um substituto da computação em nuvem, e essas duas tecnologias se

complementam.

Nesta direção, é crescente o surgimento de soluções da IoT que buscam resolver problemas de alocação de recursos. Uma promissora direção de pesquisa parte do princípio de explorar interações diretas entre coisas, com o objetivo de monitorar e controlar a si mesmos e o ambiente circundante com a mínima intervenção humana. Não obstante, abordagens autonômicas são amplamente exploradas na literatura em contextos de computação em nuvem como solução para problemas de gerenciamento de recursos (SINGH; CHANA, 2016), (SINGH; CHANA; SINGH, 2017), (GUÉROUT; ALAYA, 2013). Contudo, no âmbito da *fog Computing*, ao melhor do nosso conhecimento, não encontramos trabalhos que implementaram este tipo de abordagem.

Neste contexto, a implementação de um sistema dinâmico, extensível, dimensionável e tolerante a falhas para executar serviços IoT entre a borda da rede e a nuvem torna-se necessário uma vez que, a *fog computing* deve ser capaz de fornecer os meios para enfrentar os desafios mencionados de dados futuros e abundância de dispositivos (DASTJERDI; BUYYA, 2016).

1.4 Objetivos

Considerados os problemas e desafios apresentados, o presente trabalho busca propor uma estratégia de otimização para o problema de alocação de recursos através de um modelo de arquitetura autonômica com base no laço de controle MAPE-K.

1.4.1 Objetivos específicos

Para que nosso objetivo principal seja alcançado outros objetivos específicos devem ser concluídos, são eles:

- Identificar as principais técnicas de provisionamento de recursos no domínio da *fog computing*;
- Propor um modelo conceitual para a arquitetura que atenda aos objetivos de otimização;
- Implementar os módulos utilizando tecnologias de código aberto;
- Avaliar o método proposto utilizando ViolET (BADIGER; BAHETI; SIMMHAN, 2018), um ambiente virtual de larga escala para Internet das Coisas.

1.5 Metodologia

Esta dissertação de mestrado está caracterizada como uma pesquisa aplicada, que contém objetivos exploratórios e descritivos, com abordagem qualitativa com caráter bibliográfico e experimental. O procedimento sobre como concluir esta dissertação é dividido em três etapas.

A primeira parte desta dissertação consiste em uma revisão bibliográfica para poder analisar o estado da arte do tema proposto à partir da literatura disponível, e com isso ter um embasamento teórico necessário para o desenvolvimento deste trabalho, bem como encontrar lacunas de pesquisa. Isto inclui desenvolver um mapeamento sistemático da literatura para identificar técnicas e algoritmos utilizados para o gerenciamento de recursos em *fog computing*. A extração das informações foi feita procurando por tópicos pré-definidos mencionados no capítulo 3.

Como próximo passo, o desenho conceitual da arquitetura autonômica é feito. É digno de nota que, o desenho da arquitetura é um ponto crucial no desenvolvimento deste trabalho e, portanto, deve ser verificado e revisado durante todo o processo de desenvolvimento. Isso garante a exatidão das decisões da condução do trabalho. Durante o projeto, uma abordagem adequada de provisionamento de recursos, bem como a arquitetura de *software*, incluindo tecnologias e padrões, foram selecionados. As tecnologias foram avaliadas em relação aos requisitos funcionais e não funcionais em combinação com as dependências das ferramentas já escolhidas.

A terceira parte propõe-se a realizar uma extensa avaliação na execução de aplicações IoT no ambiente *fog computing* com o objetivo traçar um perfil da aplicação para identificar quais as limitações, sobrecarga ou subutilização do sistema distribuído, caso existam, bem como para mostrar os benefícios da estrutura implementada. O tempo de migração e disponibilidade dos serviços são utilizados como métricas de desempenho do ambiente. Além disso, a comparação da utilização da CPU, memória dos nós da rede antes e durante a orquestração do ambiente é feita. Essas métricas são utilizadas para a análise de sobrecarga causada. As métricas consideradas serão medidas através das ferramentas de monitoramento Prometheus Prometheus () e Grafana Grafana (), em um *cluster* implementado com a ferramenta Docker Swarm. Os valores obtidos são usados para comparar a necessidade de recursos para o ambiente proposto.

1.6 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, que são:

- Capítulo 1 - Introdução: Apresenta as argumentações, problemática e hipóteses sobre o tema e os objetivos, as definições preliminares de literatura;
- Capítulo 2 - Referencial Teórico: Expõe o referencial teórico desta dissertação e fornece uma visão geral sobre as tecnologias necessárias para a concepção da *fog computing*. relacionada ao tema proposto;
- Capítulo 3 - Trabalhos Relacionados: Demonstra os trabalhos correlatos com a revisão de literatura adotada (Mapeamento Sistemático), bem como são apresentados os resultados dessa revisão e a síntese do processo de sumarização dos trabalhos estudados;

- Capítulo 4 - Trabalho Proposto: Este capítulo apresenta os aspectos de concepção da abordagem proposta e são discutidas as premissas de concepção da arquitetura autonômica;
- Capítulo 5 - Resultados e Discussão: Este capítulo apresenta os resultados obtidos nesta dissertação;
- Capítulo 6 - Conclusão: Apresenta as considerações finais deste trabalho, suas contribuições, artigos publicados e linha gerais de direções e trabalhos futuros;

2

Referencial Teórico

Este capítulo aborda o referencial teórico desta dissertação e fornece uma visão geral sobre as tecnologias necessárias para a concepção da *fog computing* e, mais especificamente, para a estrutura de *fog computing*, conforme especificado no capítulo de introdução.

2.1 Internet das Coisas

O termo Internet das Coisas vem do inglês *Internet of Things* (IoT), e faz menção à capacidade que os objetos possuem de se comunicarem entre si, reportando informações acerca de seu estado e funcionamento. De acordo com Atzori, Iera e Morabito (2010) e Gubbi et al. (2013), a IoT consiste em interligar os objetos de uso cotidiano do ambiente real com a Internet, tornando-os então objetos inteligentes. A fim de esclarecer o conceito técnico de IoT, Jara, Ladid e Skarmeta (2014) afirmam que a Internet das coisas é composta, por um lado, dos chamados objetos inteligentes, isto é, dispositivos físicos pequenos e altamente restritos em termos de capacidade de memória, capacidade de computação, autonomia energética e capacidades de comunicação. Por outro lado, é composta de etiquetas de identificação e códigos que permitem identificar uma coisa específica de uma forma única e global.

Para um melhor aproveitamento dos benefícios que a IoT pode proporcionar, seu ambiente necessita de integração entre diferentes sistemas de detecção e atuação em uma única infraestrutura de serviços para permite que as aplicações se beneficiem da alta disponibilidade do serviço IoT. Assim, diferentes "coisas"inteligentes equivalentes podem fornecer serviços similares com funcionalidades comuns, mas com diferentes requisitos de QoS e custos.

Neste contexto, o paradigma da computação em nuvem é considerado como um facilitador para o desenvolvimento e implantação de plataformas IoT em grande escala, uma vez que permite o compartilhamento global de recursos em um ambiente naturalmente distribuído, provisionamento de recursos elásticos e interação flexível com clientes em nuvem (TANGANELLI;

VALLATI; MINGOZZI, 2014)(AAZAM; HUH, 2015). Além disso, a grande quantidade de conexões e interações entre os nós de borda na IoT faz dela um sistema complexo e escalável. O que pode implicar no surgimento de dificuldades para satisfazer os requisitos dinâmicos de QoS dos serviços (LIU et al., 2008), (HODGES et al., 2013).

A figura 1 ilustra as diferentes visões da Internet das Coisas abordadas por Atzori, Iera e Morabito (2010).

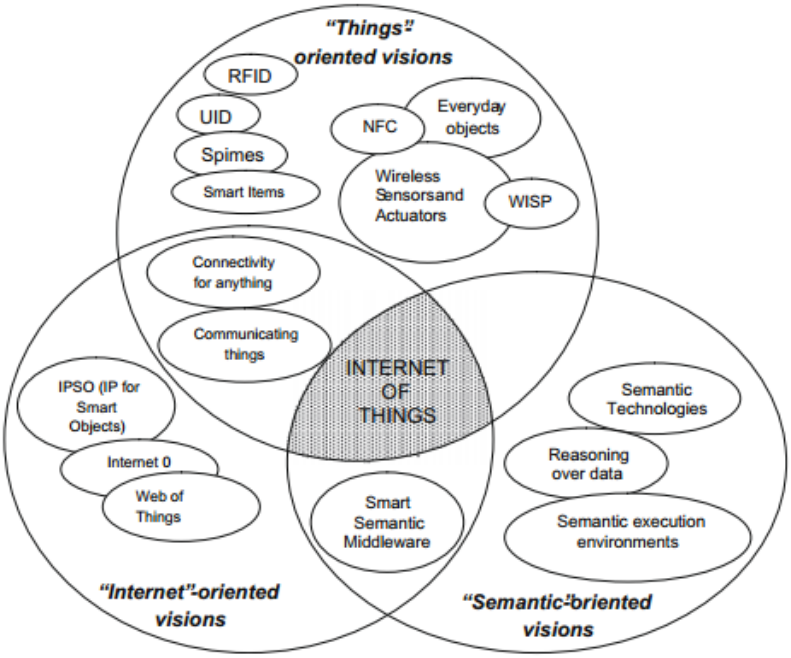


Figura 1 – Visões da Internet das Coisas. (ATZORI; IERA; MORABITO, 2010)

Essas visões destacam os principais conceitos, tecnologias e padrões que compõem a IoT, classificando-as em três grupos:

- **Visão orientada às coisas**: objetiva demonstrar propostas que assegurem o melhor aproveitamento dos recursos dos dispositivos e sua comunicação;
- **Visão orientada à semântica:** foca na representação, armazenamento, pesquisa e organização da informação gerada, procurando soluções para a modelagem das descrições que permitam um tratamento adequado para os dados produzidos pelos objetos;
- **Visão orientada à *Internet*:** tem o intuito de conceber modelos e técnicas destinadas a interoperabilidade dos dispositivos em rede.

Neste contexto, a IoT é conhecida de modo geral por seus pequenos dispositivos, com conexões amplamente distribuídas, capacidade limitada de armazenamento e processamento, o que envolve preocupações sobre confiabilidade, desempenho, segurança e privacidade.

2.2 Cloud Computing

De acordo com Mell, Grance et al. (2011), o termo *cloud computing* ou computação em nuvem é descrito como um modelo que permite acesso a uma rede sob demanda ubíqua e conveniente a um conjunto compartilhado de recursos computacionais configuráveis (ex. redes, servidores, armazenamento, aplicações e serviços). Esse paradigma separa os recursos básicos de computação das máquinas físicas individuais. Assim, esses recursos podem ser rapidamente provisionados e liberados com o mínimo de esforço para gerenciamento ou interação com o serviço do provedor.

2.2.1 Características

A computação em nuvem permite aos usuários acessar um *pool* compartilhado de recursos de computação que podem ser utilizados sob demanda em vez de restringir a uma única aplicação recursos físicos de computação, armazenamento, serviços ou rede (HASHEM et al., 2015). Entre as principais características deste paradigma podemos destacar:

- **Escalabilidade**: Para fornecer a ilusão de recursos infinitos, mais máquinas virtuais podem ser provisionadas rapidamente no caso de pico de demanda, e liberadas rapidamente para acompanhar a demanda em qualquer quantidade e a qualquer momento. Esses métodos de dimensionamento podem ser feitos automaticamente de acordo com as condições predefinidas do usuário ou pelo provedor de serviço.
- **Dimensionamento rápido**: o modelo de computação em nuvem permite às empresas aumentar ou diminuir a expansão quase instantaneamente, com base na demanda atual, eliminando gargalos e ativos subutilizados.
- **Facilidade de uso e manutenção**: com a computação em nuvem, recursos e atualizações podem ser implantados de forma automatizada e padronizada, aumentando a acessibilidade e eliminando inconsistências e a necessidade de atualizações manuais.
- **Melhor recuperação de desastres**: as soluções de *backup* baseadas na nuvem tendem a ser relativamente baratas e fáceis de usar. O modelo de computação em nuvem também significa que arquivos importantes não ficam presos a máquinas individuais, as quais, inevitavelmente, falham em algum momento.
- **Redução de custos**: Esta característica corresponde ao fato de pagar apenas pelos recursos que se deseja utilizar, e por um curto período de tempo. Isso significa que é possível cortar custos e gerir recursos de forma enxuta e otimizada, proporcionando outra grande vantagem para as pequenas e médias empresas. Desta forma, o cliente não precisa lidar com a gestão de recursos inativos. Isso ocorre, por exemplo, com o licenciamento de *softwares* pelo modelo SaaS (*Software as a Service*, ou *Software* como Serviço, em português).

Assim, a companhia pagará apenas pelo número de licenças ativas no período de validade do contrato, o que torna os gastos mais precisos e, em geral, menores.

2.2.2 Modelos de Serviço

Com relação aos serviços de computação em nuvem, existem diversos modelos disponíveis (MACHADO; MORENO; RIBEIRO, 2017). Os 3 principais modelos de oferta são apresentados a seguir: *software* como serviço (SaaS - *software as a Service*), Plataforma como Serviço (PaaS - *Platform as a Service*) e Infraestrutura como Serviço (IaaS - *Infrastructure as a Service*). Abaixo uma breve descrição de cada modelo de serviço.

- **SaaS**: Refere-se ao modelo de fornecimento de aplicações sob demanda através da Internet. Exemplos de fornecedores de SaaS incluem Google Docs, Gmail, Salesforce.com e Online Payroll.
- **PaaS**: Este modelo refere-se ao fornecimento de recursos da camada de plataforma, incluindo suporte ao sistema operacional e de desenvolvimento de *software*. O PaaS é voltado para empresas desenvolvedoras de software, que pretendem criar um ambiente de execução de sistemas próprios na nuvem, vendendo algum serviço ou solução SaaS. Esse é o caso de plataformas de vendas e ferramentas de desenvolvimento.
- **IaaS**: é um modelo de negócios em que empresas fornecem o acesso aos recursos básicos de um servidor, podendo ser contratado pelo cliente final. Ele pode incluir mas não se limitar, a contratação de serviços de rede, recursos computacionais ou de armazenamento de dados. Este é um modelo onde a utilização de um *software* não está relacionado a compra de licenças, ou seja, você utiliza algum *software* e paga por sua utilização. Como exemplo podemos citar, Flexiscale e Amazon EC2.

2.2.3 Modelos de Implantação

Os modelos de serviço apresentados anteriormente estão hospedados em diferentes modelos de implantação. Do ponto de vista dos modelos de implantação, a Computação em Nuvem se classifica em diferentes vertentes, dentre as quais de acordo com o NIST (MELL; GRANCE et al., 2011) destacam-se: Nuvens Privadas, Públicas, Comunitárias e Híbridas.

- **Nuvens Privadas**: oferecem a ideia de fornecer serviços para a própria organização, sendo operadas e utilizadas apenas pela mesma. Existem ainda duas variantes da Nuvem Privada, a *On-premise* e a *Off-premise* (ou Hospedada Externamente). Na *on-premise* a infraestrutura de nuvem é implantada por uma organização em seus *data centers*, dentro de suas instalações, possibilitando desta forma o controle completo sobre a infraestrutura e os dados. Já na *off-premise* a implementação da nuvem é terceirizada para um provedor

externo com a nuvem hospedada em instalações de um prestador e os consumidores se conectam a ele através de uma rede segura, onde as políticas de acesso isolam os recursos de nuvem de outros inquilinos.

- **Nuvens Públicas**: comportam um modelo que disponibiliza ambientes para o público em geral e são normalmente comercializadas por corporações com grande poder de armazenamento e processamento.
- **Nuvens Comunitárias**: baseiam-se um ambiente de computação em nuvem compartilhado entre organizações com interesses em comum. Da mesma forma, e conceito, que a nuvem privada, as nuvens comunitárias podem ser *on-premise* ou *off-premise*.
- **Nuvens Hibridas**: tratam da composição entre dois ou mais ambientes de estruturas distintas, como exemplo podemos citar a junção de nuvens públicas e privadas gerando uma única nuvem.

2.3 Fog Computing

Com o passar dos anos a computação em nuvem se consolidou como a principal opção para prover recursos computacionais e de armazenamento para dispositivos e aplicações IoT. A comunicação entre estes dois paradigmas permite que a demanda computacional seja atendida através de *data centers* localizados geograficamente em diferentes pontos espalhados pelo mundo. Contudo, um problema comum na computação em nuvem é a latência. Uma vez que as aplicações podem estar disponíveis para dispositivos distribuídos geograficamente, a conexão com os *data centers* pode depender de fatores como por exemplo, localização e largura de banda. Isto também afeta diretamente aplicações de tempo real. Para superar estes desafios, Bonomi et al. (2012) propuseram um novo paradigma chamado *fog computing* ou computação em neblina.

2.3.1 Características

Fog computing é um paradigma inovador que realiza computação distribuída, serviços de rede e armazenamento, além da comunicação entre *data centers* na nuvem até os dispositivos ao longo da borda da rede (MACHADO; MORENO; RIBEIRO, 2017). A seguir são apresentadas as suas principais características.

- **Heterogeneidade**: A *fog computing* é uma plataforma virtualizada que oferece serviços computacionais, de rede e de armazenamento entre a computação em nuvem e dispositivos finais de diferentes tipos e formas.
- **Distribuição geográfica**: A *fog computing* possui uma implementação amplamente distribuída para oferecer serviços de alta qualidade para dispositivos finais móveis e fixos.

- **Localização de borda, percepção de localização e baixa latência**: O conceito *fog computing* foi implementado para suprir a falta de suporte para pontos finais, com serviços de qualidade à beira da rede.
- **Interação em tempo real:** Diversas aplicações de *fog computing*, como sistemas de monitoramento de tráfego, exigem processamento em tempo real em vez de processamento em lote.
- **Suporte a mobilidade**: O suporte a mobilidade é essencial para muitas aplicações em ambientes *fog computing*, para permitir a comunicação direta com dispositivos móveis usando protocolos como o protocolo de separação de localização / ID da Cisco, que desacopla a identidade do *host* da identidade de localização usando um sistema de diretório distribuído.
- **Grande escala de redes de sensores**: Isso é aplicável ao monitorar o ambiente ou em rede inteligente, usando sistemas inerentemente distribuídos que requerem computação distribuída ou recursos de armazenamento.
- **Prevalente para acesso sem fio:** A maioria dos pontos de acesso sem fio e o *gateway* de dispositivos móveis são exemplos típicos de um nó de *fog computing* na rede.
- **Interoperabilidade**: Os componentes de *fog computing* devem ser capazes de interoperar para garantir suporte para ampla gama de serviços, como transmissão de dados.

Segundo Peralta et al. (2017), as características da *fog computing* as torna uma boa solução para uma ampla gama de aplicações, tais como, casas inteligentes, saúde, veículos inteligentes conectados (SVC), e análises de *big data*, incluindo também as aplicações relacionadas à Indústria 4.0. A seguir apresentamos com mais detalhes os tipos de aplicação sugeridas na literatura que podem se beneficiar com a adoção da *fog computing*.

2.3.2 Aplicações IoT em Fog Computing

De acordo com Osanaiye et al. (2017), diferentes aplicações que são suportadas pela *fog computing* foram sugeridas na literatura. As categorias das aplicações são divididas em: (i) Aplicações em tempo real; (ii) Aplicações quase em tempo real; (iii) Aplicações introduzidas em redes.

- (i) As aplicações em tempo real são aplicações de baixa latência, que funcionam dentro de um período de tempo pré-definido, sendo classificada pelo usuário como imediata ou urgente. Entre os tipos de aplicação, podemos citar video *streaming* (HONG et al., 2013), jogos (WANG; DEY, 2012) e *healthcare* (GIA et al., 2015) e semáforos inteligentes (STOJMENOVIC; WEN, 2014).

- (ii) Quase em tempo real, por outro lado, são aplicações que estão sujeitas a atraso de tempo introduzido pelo processamento de dados ou transmissão de rede, entre o momento em que ocorre um evento e o uso dos dados para processamento. Podemos citar como exemplo, *smart grids*, *smart cities* e *smart vehicles* (LU et al., 2014).

- (iii) A *fog computing* também pode ser introduzida em uma rede para aplicações que não necessitem de processamento e transmissão em tempo real com o objetivo de reduzir a quantidade de tráfego no núcleo de processamento.

Por outro lado, de acordo com Kopetz (2011) , aplicações de tempo real são aquelas as quais o tempo de resposta é de fundamental importância para a obtenção do resultado esperado. Para este fim, faz-se necessário um gerenciamento eficiente de recursos para realizar uma seleção adequada de coisas que correspondam às requisições de aplicações, garantindo ao mesmo tempo atender aos respectivos requisitos de QoS.

2.4 Provisionamento de Recursos

O provisionamento de recursos é um tópico crucial em diversas áreas de pesquisa. De certo modo, as consequências do avanço tecnológico evidenciam a clara a necessidade da utilização de recursos disponíveis da maneira mais eficiente possível. Neste contexto, a eficiência do provisionamento está vinculada a otimização de vários objetivos, tais como, otimização de custo, energia, tempo de execução e utilização de recursos. De acordo com Bonomi et al. (2014), o provisionamento de recursos é o procedimento para orquestrar, alocar, desalocar e monitorar recursos do sistema disponíveis. Essas ações mencionadas são cruciais para permitir um gerenciamento de recursos de forma eficiente e ciente de QoS, ou seja, um provisionamento dos recursos.

Portanto, não apenas a demanda dependente do tempo precisa ser considerada, mas também as métricas de QoS precisam ser monitoradas e cruzadas em tempo real com os SLAs. Dependendo da variação da carga de trabalho do sistema, o procedimento de provisionamento de recursos muda em complexidade uma vez que os recursos precisam ser adaptados cada vez mais (KALYVIANAKI, 2009).

Nos parágrafos seguintes, são descritos os aspectos mais importantes do provisionamento de recursos no modelo de arquitetura *fog computing* (GUPTA et al., 2016).

- **Componentes de monitoramento:** acompanham a utilização dos recursos e a disponibilidade de sensores, atuadores, dispositivos de *fog computing* e elementos de rede. Eles acompanham os aplicativos e serviços implantados na infraestrutura, monitorando seu desempenho e status fornecendo essas informações a outros serviços, conforme necessário.

- **Gerenciamento de recursos:** é o componente central da arquitetura e consiste em componentes que gerenciam recursos consistentemente de forma que as restrições de QoS de nível de aplicação sejam atendidas e o desperdício de recursos seja minimizado. Para este fim, os componentes do provisionamento desempenham um papel importante ao acompanhar os recursos disponíveis através de informações fornecidas pelo serviço de monitoramento para identificar os melhores candidatos para hospedar um módulo de aplicação. Este aspecto é descrito como orquestração, ou seja, um processo onde as informações de monitoramento do sistema são analisadas e um plano de provisionamento de recursos é calculado. O processo de orquestração resulta em plano de provisionamento de recursos que define onde os serviços específicos são implementados e em quais serviços as solicitações de tarefas recebidas são processadas.

- **Monitoramento de energia:** um dos desafios mais difíceis e que a maioria das soluções IoT enfrenta é o uso de recursos dos nós IoT considerando questões sobre o consumo de energia. Em contraste com os *data centers* da nuvem, a *fog computing* engloba um grande número de dispositivos com consumo de energia heterogêneo, dificultando a gestão de energia. Portanto, avaliar o impacto das aplicações e políticas de gerenciamento de recursos no consumo de energia é crucial antes da implantação em ambientes de produção.

De acordo com Kalyvianaki (2009), o provisionamento de recursos pode ser feito de forma proativa ou reativa. Um cenário proativo implica verificação periódica do status do sistema e, de acordo com os dados monitorados, o provisionamento de recursos é feito. Frequentemente, mecanismos preditivos são aplicados para prever demandas futuras de recursos e agir antes que o sistema necessite. Por outro lado, um cenário de provisionamento reativo, lida diretamente com eventos como falhas, eventos de alerta, etc. Por exemplo, um componente de monitoramento pode gerar um evento indicando que há uma sobrecarga no ambiente, ou seja, uma utilização específica de um recurso computacional excede um limite definido e o sistema reage com uma ação com o objetivo de contornar o problema apresentado.

Além disso, o procedimento de provisionamento de recursos pode ser feito de acordo com várias abordagens diferentes a depender do objetivo da otimização. Em um cenário em que o objetivo é otimizar o consumo de energia, os serviços podem ser, por exemplo, distribuídos de modo a maximizar a utilização dos nós, ou seja, alocar o máximo possível de serviços no mínimo de nós. Por outro lado, se o objetivo for otimizar a latência, o ambiente pode priorizar a alocação dos serviços nos nós mais próximos geograficamente dos usuários.

O problema de otimização pode ser formulado em termos de programação dinâmica, especificamente programação linear (DANTZIG, 2016), e pode ser resolvido por métodos matemáticos exatos, ou por algoritmos heurísticos.

Heurísticas são algoritmos aproximados a problemas do mundo real, que buscam soluções quase ótimas. Para melhorar a solução de heurística, muitas vezes o cenário e o ambiente

do problema precisam ser refinados e descritos com mais detalhes. Exemplos de algoritmos heurísticos são algoritmos gulosos e busca local (LUKE, 2017).

2.5 Computação Autonômica

A IBM propôs a arquitetura de computação autonômica (COMPUTING et al., 2006), que abrange vários recursos de gerenciamento, incluindo auto-configuração, auto-cura, auto-otimização e auto-proteção. Essas características exigem que o sistema conheça o contexto em que suas entidades distribuídas evoluem. Habilitar a consciência do contexto em um ambiente heterogêneo requer um modelo adequado de arquitetura e gerenciamento de conhecimento que abstraia da melhor maneira a heterogeneidade, a distribuição e a dinâmica de entidades de ambiente pervasivos.

Diferentes termos são relacionados com a computação autonômica, tal como o termo sistemas auto-gerenciados (self-management) abordados por Kramer e Magee (2007), sistema autonômicos descrito em Sterritt (2005). Além disso, os desafios mencionados para sistemas distribuídos também são abordados por uma área de pesquisa chamada Computação Orgânica (HUEBSCHER; MCCANN, 2008). Autoadaptação *(self-adaptation)* ou *softwares* autoadaptativos são outros termos muito utilizados que, de acordo com Huebscher e McCann (2008), esses possuem um domínio mais limitado que a computação autonômica, estando incluídos nela. Apesar de nomes diferentes,os conceitos dos domínios citados são extremamente relacionados e, em muitos casos, eles podem ser usados como sinônimos (SALEHIE; TAHVILDARI, 2009). O caso das propriedades auto-* e os laços de controle são os elementos mais comuns abordados por esses tipos de trabalhos, ambos apresentados nas próximas subseções. Uma das grandes motivações para a construção desse tipo de sistemas consiste na busca para eliminar ou ao menos diminuir a necessidade de intervenção humana perante o *software*, tendo como consequência a redução de custos (HUEBSCHER; MCCANN, 2008).

2.5.1 Propriedades Auto-*

As propriedades auto-* representam as propriedades que delimitam os objetivos de adaptação aplicados em um sistema que o tornam um sistema autoadaptativo. De acordo com Kephart e Chess (2003), essas propriedades foram definidas baseadas no mecanismo biológico de autoadaptação, sendo elas:

- **Auto-configuração *(self-configuring)***: Confere a capacidade de se reconfigurar automaticamente e dinamicamente em respostas às mudanças, instalando, atualizando e integrando, além de compor e decompor entidades de *software*.
- **Auto-cura *(self-healing)***: Refere-se à capacidade autoadaptativa de detectar, diagnosticar, evitar e reparar problemas resultantes de *bug*s, defeitos ou falhas no *software* e *hardware*.

- **Auto-otimização *(self-optimising)***: *software* autoadaptativos com essa capacidade buscam continuamente formas de melhorar seu funcionamento, e se tornarem mais eficientes no custo, desempenho ou QoS para satisfazer os objetivos pré-definidos, que podem ser além de centrados no *software*, também centrados no usuário.
- **Auto-proteção *(self-protecting)***: Indica que o *software* vai buscar se proteger de ataques maliciosos e de falhas.

Embora essas quatro propriedades auto-* representem a maioria dos possíveis usos em um sistema autoadaptativo, diversos autores propuseram outras propriedades de forma mais específica ou mesmo equivalentes. Este trabalho ficará restrito à propriedade da Auto-otimização.

2.5.2 Laço de controle

O laço de controle MAPE-K foi introduzido pela IBM em Computing et al. (2006) e em seguida discutido no contexto de sistemas adaptativos por Brun et al. (2009) e Kephart e Chess (2003). Esta estratégia proporciona a habilidade de um ambiente computacional se auto-gerenciar e dinamicamente adaptar-se a mudanças de acordo com objetivos e políticas de negócios. A maioria dos trabalhos sobre autoadaptação segue o padrão comum de laço fechado: Monitorar, Analisar, Planejar e Executar, ou MAPE, conectados por um *feedback* (NALLUR; BAHSOON, 2013).

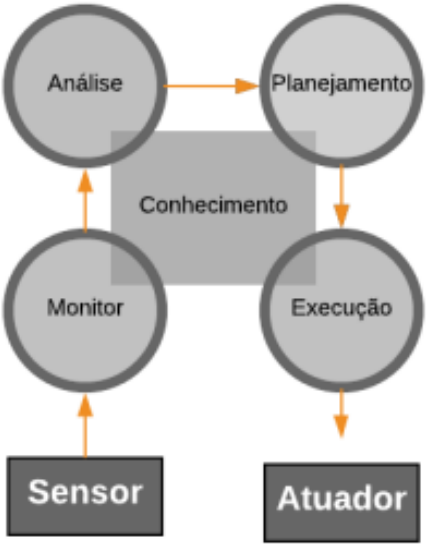


Figura 2 – Gerenciador autonômico. Adaptado de Computing et al. (2006)

A função de cada um dos componentes é descrita a seguir:

- **Sensores:** São entidades de *software* que geram uma coleção de dados refletindo o estado do sistema. Por exemplo, a latência na comunicação com serviços, o consumo de energia e

a informação de desconexão são informações monitoradas por sensores de um dispositivo móvel;

- **Atuadores:** Realizam as ações de adaptação que podem ser, por exemplo, desde uma simples alteração de um parâmetro do elemento autonômico, até como configurar uma nova instância de serviço em um agrupamento de servidores de *web* o qual envolve a alocação e gerenciamento de diversos recursos;
- **Monitor:** Esta função provê os mecanismos para coletar, agregar, filtrar e reportar detalhes (como métricas e topologias) coletados dos recursos gerenciados;
- **Análise:** A função de análise provê os mecanismos que correlacionam e modelam situações complexas (por exemplo, previsão de séries temporais e modelos de filas). Esses mecanismos permitem que o gerente autonômico aprenda sobre o ambiente e ajude a predizer situações futuras;
- **Planejamento**: Provê os mecanismos que constroem as ações necessárias para alcançar as metas e objetivos. Essa etapa utiliza políticas para guiar a tarefa.
- **Execução:** Provê os mecanismos que controlam a execução do plano considerando atualizações dinâmicas;
- **Conhecimento**: Contém os dados usados pelas quatro funções autonômicas do gerente, as quais armazenam e acessam o conhecimento de forma compartilhada. Esse conhecimento pode incluir dados tais como informação sobre topologia, histórico de *logs*, sintomas e políticas de gerenciamento.

3

Trabalhos Relacionados

3.1 Metodologia

Este trabalho aplicou as diretrizes da (SLM, *Systematic Literature Mapping*), abordadas em Petersen, Vakkalanka e Kuzniarz (2015), para pesquisar, selecionar, revisar e sintetizar os desafios da *fog computing* de publicações acadêmicas relevantes entre os anos de 2012 e 2018. Seguindo a SLM, esta seção traz os passos necessários para condução deste estudo. Para tanto, faz-se necessário a apresentação do objetivo, questões de pesquisa, da estratégia de busca e da metodologia utilizada para a seleção dos artigos de estudos primários. Além disso, cada subseção seguinte detalha cada um dos passos no processo de mapeamento sistemático realizado no presente estudo.

3.1.1 Objetivo

Esta pesquisa tem como objetivo analisar publicações científicas com o propósito de identificar quais contribuições foram propostas e suas limitações no âmbito do paradigma *fog computing* em relação a abordagens para o gerenciamento de recursos que beneficiem a utilização de aplicações sensíveis à latência, do ponto de vista de pesquisadores, no contexto da IoT.

3.1.2 Questões de Pesquisa

De acordo com Petersen, Vakkalanka e Kuzniarz (2015), a questão de pesquisa é primeira etapa da fase de planejamento, onde deve-se determinar o que se está procurando e definir quais resultados se deve alcançar com o mapeamento sistemático.

As questões apresentadas a seguir foram definidas para nortear a pesquisa e traçar um perfil das publicações existentes na literatura especializada, a saber:

(Q1) Quais as técnicas mais utilizadas para a alocação de recursos em um ambiente de *fog computing*?

(Q2) Quais as ferramentas de simulação são mais utilizadas?

(Q3) Quais as plataformas de hardware são mais utilizadas?

(Q4) Quais são as métricas utilizadas para avaliar a abordagem?

(Q5) Quais os tipos de domínio de aplicações podem se beneficiar com a implantação da *fog computing?*

Essas perguntas são a base para a construção da *string* de busca e para a definição dos critérios de inclusão e exclusão, servindo para especificar os pontos a serem observados e ponderados na condução da pesquisa, visando a obtenção de resultados que sejam como um resposta a essas questões.

3.1.3 Escopo e Restrições da Pesquisa

Com o objetivo de assegurar a viabilidade da pesquisa, foi definido um escopo para a mesma, que pode ser descrito por meio da definição de critérios de seleção de fontes e algumas restrições. Para a seleção das fontes de pesquisa, foram definidos os seguintes critérios:

- Disponibilidade para consultas web;
- Disponibilidade através do portal de periódicos da Capes (https://www.periodicos.capes.gov.br).
- Disponibilidade de artigos na íntegra por meio do domínio da UFS/IFS ou a partir da utilização da engine de busca *Google* e/ou *Google Scholar*;
- Disponibilidade de artigos em inglês, uma vez que é o idioma adotado pela grande maioria das conferências e periódicos nacionais e internacionais relacionados com tema de pesquisa.

3.1.4 Seleção de Fontes

Para a realização da busca por estudos relevantes, as bases eletrônicas alvo desse estudo foram as descritas a seguir na tabela 1:

Tabela 1 – Bases Eletrônicas

Base	**URL**
Scopus	http://www.scopus.com
IEEEXplore	http://ieeexplore.ieee.org
ScienceDirect	http://www.sciencedirect.com
Springer	https://www.springer.com/
ACM	https://dl.acm.org/

3.1.5 Identificação de Palavras-Chave e Sinônimos

Baseado na questão de pesquisa, três principais palava-chave são inicialmente identificadas, a saber, *Internet of Things*, *Fog Computing* e *Resource Management*. Além disso, possíveis variações como sinônimos ou formas do singular/plural são consideradas.

Como forma de abranger sinônimos (e com base em alguns artigos encontrados previamente), foram adicionadas as palavras: *edge computing*, como forma de sinônimos para *fog Computing*; *orchestration*, *service placement*, *resource provisioning* e *optimization* como forma de sinônimos para *resource management*.

3.1.6 Criação da String de Busca

A *string* de busca é um procedimento que deve ser adaptado para os motores de busca específicos, com o objetivo de produzir um retorno mais aproximado do ideal para a pesquisa. Uma *string* ineficaz pode trazer um grande número de falsos positivos (PETERSEN; VAKKALANKA; KUZNIARZ, 2015). O resultado da *string* é apresentado a seguir:

((fog AND computing OR fog OR edge AND computing OR edge) AND (internet AND of AND things OR iot) AND (resource AND management OR placement AND service OR quality AND of AND service OR qos OR orchestration OR application OR resource AND provisioning OR optimization OR optimisation))

As principais palavras-chaves foram conectadas usando o operador lógico AND. Por sua vez, as possíveis variações e sinônimos foram conectados usando o operador lógico OR.

3.1.7 Seleção dos Estudos Primários

De acordo com Kitchenham (2004), devem ser seguidos critérios de inclusão e exclusão para os artigos que são retornados pela *string* de busca. Além disso, os critérios de inclusão e exclusão dos estudos primários são os que vão nortear os pesquisadores na seleção dos estudos que foram coletados das fontes de pesquisas, além do que determina o rigor da pesquisa e impossibilita os viesses dos pesquisadores no momento da seleção. Foram definidos os seguintes critérios de inclusão:

- CI1 - Selecionar publicações que apresentam modelos de arquitetura, técnicas ou métodos aplicados a gerenciamento de serviços em ambientes *fog computing*;
- CI2 - Selecionar publicações que descrevem a implementação de técnicas ou métodos aplicados a gerenciamento de recursos em ambientes *fog computing*;

Em paralelo aos critérios de inclusão, foram definidos critérios de exclusão, descritos a seguir:

- CE1 - Não selecionar publicações que não satisfaçam a nenhum critério de inclusão;
- CE2 - Não selecionar publicações em que o idioma seja diferente do exigido;
- CE3 - Não selecionar publicações de artigos duplicados;
- CE4 - Não selecionar publicações em que abordagem voltada para o gerenciamento de recursos não seja em um ambiente *fog computing*.
- CE5 - Não selecionar publicações em que o conteúdo disponha apenas conceitos.

3.1.8 Processo de Seleção Preliminar (1° Filtro)

Foram selecionados artigos que apresentaram informações no título e no *abstract* relacionados à questão de pesquisa principal.

3.1.9 Processo de Seleção Final (2° Filtro)

Como a leitura de duas informações (título e *abstract*), não são suficientes para identificar se o estudo é realmente relevante para a pesquisa realizada, torna-se necessário realizar a leitura completa dos estudos que restaram do 1° filtro. Dessa forma, esta fase do mapeamento, tem como objetivo fazer uma análise mais apurada dos estudos, identificando e extraindo dados também de acordo com os critérios de inclusão e exclusão descritos anteriormente. A tabela 2 apresenta o resultado do critério de seleção para cada base de pesquisa utilizada.

Tabela 2 – Total de Artigos Selecionados com Critérios Aplicados em Cada Base

Base	**Resultado da pesquisa**	**Resultado do 1º filtro**	**Resultado do 2º filtro**
Scopus	393	129	5
IEEEXplore	227	88	11
ScienceDirect	247	97	5
Springer	189	135	8
ACM	365	102	4
Total	1.421	551	33

Utilizando os 33 trabalhos selecionados após aplicados os critérios de inclusão e exclusão, foi feita a análise detalhada de cada artigo para posterior extração de informações que possam responder às questões de pesquisa propostas e consequentemente, alcançar o objetivo do presente trabalho.

3.2 Análise dos resultados

Nesta subseção é apresentada a análise dos resultados obtidos a partir da extração de informações dos 33 trabalhos para responder às questões de pesquisa do presente estudo. A seguir são apresentados os resultados relacionando-os com as questões de pesquisa apresentadas na subseção 3.1.2 deste trabalho.

3.2.1 Quais as técnicas mais utilizadas para a alocação de recursos em um ambiente de *fog computing* (Q1)?

Esta questão visa apontar lacunas de pesquisa e como analisar tendências da utilização de técnicas de alocação de recursos em *fog computing*. Classificamos os trabalhos analisados em duas perspectivas, provisionamento e escalonamento de recursos. A primeira perspectiva está relacionada a onde alocar e a segunda corresponde a quando e como alocar os recursos. Os resultados são apresentados a seguir.

3.2.1.1 Provisionamento de Recursos

De acordo Singh, Chana e Singh (2017), provisionamento de recursos é um processo que busca identificar quais recursos são adequados para uma determinada demanda de carga de trabalho, baseada nas necessidades de QoS descritas pelo usuário. Podemos considerar que um dos seus objetivos é tentar solucionar o problema de como e onde alocar recursos entre os dispositivos disponíveis no ambiente. Neste contexto, a eficiência está vinculada a vários objetivos, por exemplo, otimização de custo, energia, tempo de execução e utilização de recursos. Assim, definição de melhor execução pode variar de acordo com o modelo do sistema e o objetivo de cada pesquisa. A seguir apresentamos relevantes trabalhos que apresentam soluções para resolver o problema de provisionamento em um ambiente *fog computing*.

Skarlat et al. (2017a) propuseram um algoritmo genético como solução para alocação de serviços entre ambientes de nuvem e *fog computing*. Além disso, os autores apresentam o conceito de *fog colonies*, que subdivide o ambiente em pequenos grupos que são orquestrados por dispositivos *fog* com mais poder computacional e permite que tarefas sejam alocadas em colônias vizinhas caso necessitem.

Cardellini et al. (2015), desenvolveram um modelo matemático para o problema de distribuição de aplicações DSP que considera atributos de aplicações e recursos disponíveis para flexibilizar parâmetros de qualidade de serviço. Além disso, os autores propuseram um protótipo baseado no *Apache Storm* como uma ferramenta para a comparação de diferentes políticas de alocação de recursos.

Taneja e Davy (2017) apresentaram uma heurística para mapear e alocar módulos de aplicações IoT entre o ambiente de nuvem e dispositivos na borda da rede. O algoritmo utiliza

um estratégia gulosa que ordena de forma ascendente nós e módulos, e os alocam de acordo com as restrições de recursos computacionais.

Aazam e Huh (2015) apresentaram um modelo de gerenciamento de recursos orientado a serviços. A abordagem considera características do usuário para estimar valores para o provimento de serviços.

3.2.1.2 Escalonamento de Recursos

De acordo com Singh, Chana e Singh (2017), escalonamento de recursos é o processo de mapear e executar a carga de trabalho solicitada de acordo com os recursos selecionados pelo provisionador de recursos. Nesta direção, considerando técnicas de como os serviços e aplicações, podem ser movidos durante a execução e qual o melhor local para eles serem executados.

Velasquez et al. (2017), propuseram uma arquitetura para alocação de serviços de aplicações IoT entre a nuvem e os dispositivos na rede. O principal objetivo desta arquitetura é localizar em quais nós os serviços estão alocados e convenientemente migrá-los para outros nós de acordo com as condições da rede.

Tärneberg et al. (2017) investigaram o posicionamento orientado a aplicações e apresentam um modelo de sistema para rede de nuvens móveis com uma heurística de posicionamento dinâmico com o objetivo de garantir o desempenho de aplicações, minimizar o custo e resolver problemas de assimetria de recursos.

Urgaonkar et al. (2015) discutiram uma abordagem baseada na técnica de otimização Lyapunov com o objetivo de resolver o problema de migração de serviços, atendendo aos desafios de mobilidade dos usuários e variação de demanda, preservando qualidade de serviço e performance.

Plachy, Becvar e Strinati (2016) apresentam um algoritmo de colocação de VM cooperativo e dinâmico, associado a outro algoritmo cooperativo para selecionar um caminho de comunicação adequado. Eles usam a migração de VMs para resolver problemas de mobilidade do usuário.

Em relação ao gerenciamento de recursos para tarefas de migração, Rodrigues et al. (2017) apresentaram um estudo com foco na migração de VMs, mas com diferentes objetivos de otimização tais como, priorizar o uso de energia verde e minimizar tempos de resposta.

Yousaf e Taleb (2016) propuseram um sistema de migração e gerenciamento de VMs que leva em conta a relação entre as unidades de recursos ao tomar decisões de migração. Eles apresentam esse trabalho no contexto de redes 5G, mas a técnica pode ser aplicada em outros domínios.

Kaur et al. (2017) propuseram uma arquitetura que utiliza métodos de seleção e escalonamento de tarefas, com o objetivo de reduzir o consumo de energia ao mesmo tempo em que

mantém níveis aceitáveis de SLA (*Service Level Agreement*) (LEITNER et al., 2012) .

No trabalho proposto por Nassiffe et al. (2016), os autores apresentaram um mecanismo utilizando métodos de ponto interno como meio para otimizar o QoS de sistemas de tempo real sujeitos a restrições de energia e escalonamento.

O trabalho apresentado por Suto et al. (2015), propôs um esquema que controla o tempo de desligamento das antenas e a conectividade de rede para reduzir o consumo de energia do sistema, enquanto satisfazem níveis aceitáveis de SLA.

Outro aspecto importante a ser considerado são os algoritmos aplicados para este tipo de problema. Os principais trabalhos apresentam soluções baseadas em heurísticas, programação linear e abordagens evolucionárias.

Xu et al. (2014), propuseram um modelo para o problema de alocação de tarefas na borda da rede. Baseado neste modelo, os autores apresentam um mecanismo que maximiza a utilização de recursos entre a borda da rede e os provedores de serviço.

Bajpai, Choudhury e Choudhury (2017) propuseram um mecanismo de otimização para alocar serviços entre o núcleo e borda da rede, de tal forma que o custo de fornecer um serviço ao consumidor seja minimizado sem violar os requisitos de QoS dos consumidores. Além disso os autores, modelaram o custo do consumo de serviços para a nuvem em termos de disponibilidade do serviço, em diferentes níveis e custos de replicação nos níveis correspondentes. A estratégia apresentada permite tomar uma decisão sobre quando e onde um determinado serviço deve ser replicado.

Mennes et al. (2016), apresentaram GRECO (*Genetic Algorithm for Reliable Application Placement in Hybrid Clouds*), um algoritmo genético como solução para um problema de alocação de recursos e o compararam com uma solução de programação de inteiro linear.

Wang et al. (2017) , propuseram um algoritmo online baseado em uma heurística gulosa para solucionar o problema de alocação de aplicações, com o objetivo de minimizar a utilização máxima de recursos em nós e links físicos entre dispositivos na borda da rede.

Em He et al. (2016), os autores propuseram um algoritmo heurístico de otimização baseado no enxame de partículas MPSO-CO. Que realiza o balanceamento de carga entre os nós na borda da rede utilizando redes SDN.

Dentre os principais abordagens utilizadas nos trabalhos elencados deste mapeamento, podemos agrupar os principais algoritmos como solução baseada em heurísticas, abordagens evolucionárias, soluções baseadas em programação linear e soluções para problemas conhecidos.

Heurísticas: São algoritmos próximos a problemas do mundo real, que buscam soluções quase ótimas. Para melhorar a solução de heurística, muitas vezes o cenário e o ambiente do problema precisam ser refinados e descritos com mais detalhes. Exemplos de trabalhos que abordam algoritmos heurísticos são Taneja e Davy (2017),Skarlat et al. (2017b), Habak et al.

(2017), Tärneberg et al. (2017), Zamani et al. (2017), Wang et al. (2017).

Programação Linear: Outro grupo de trabalhos resolvem esse problema através de uma abordagem analítica, a programação linear. Por exemplo, Cardellini et al. (2015), Velasquez et al. (2017), Liu, Shinkuma e Takahashi (2014), Liu, Lee e Zheng (2016).

Abordagens Evolucionárias: Outra abordagem para este problema é usar um algoritmo genético, técnica bem consolidada em ambientes de nuvens tradicionais. Contudo, no contexto da *fog computing* poucos trabalhos, um exemplo é o de Skarlat et al. (2017a), Mennes et al. (2016).

A figura 3 ilustra os tipos de algoritmos em relação ao número de trabalhos por tipo de algoritmo, seguido por um quadro que apresenta as referências dos trabalhos onde estes algoritmos são utilizados.

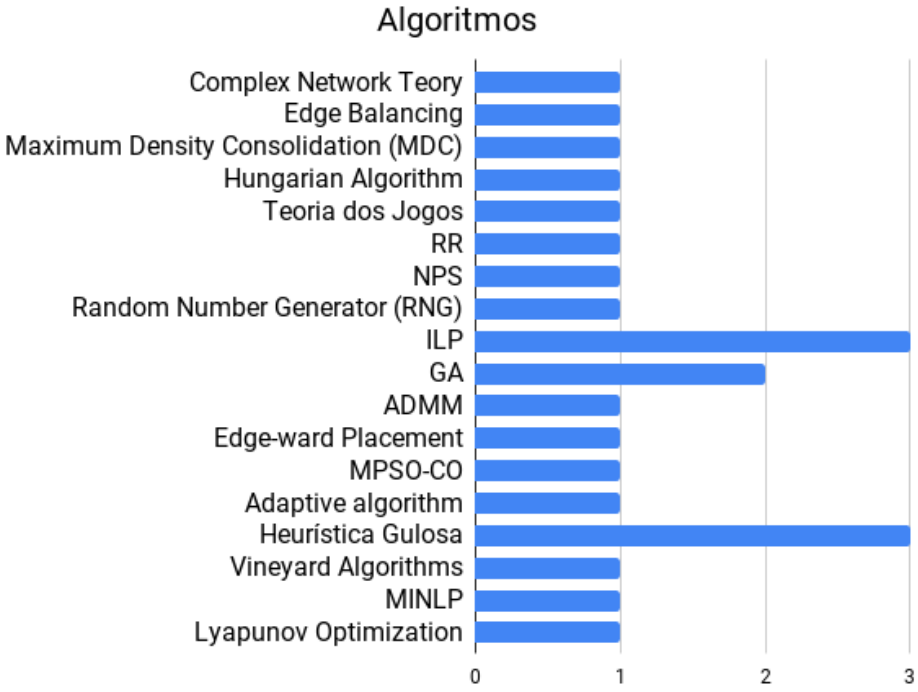


Figura 3 – Algoritmos

Também podemos observar através do resumo apresentado na tabela 3, que a maioria dos trabalhos utilizam programação linear para a solução dos problemas de otimização, seguido de heurísticas e abordagens evolucionárias. O uso de algoritmos genéticos (GA) (MORELL; ALBA, 2018), já é um tema consolidado na área de computação em nuvem, contudo, em ambientes *fog computing* ainda são poucos explorados.

No tópico a seguir, apresentamos os resultados para a segunda questão de pesquisa.

Tabela 3 – Algoritmos Analisados nos Trabalhos Selecionados

Trabalho	Algoritmos
(SUTO et al., 2015)	Complex Network Teory
(VERMA et al., 2016)	Edge Balancing
(POORANIAN et al., 2017)	Modified Best Fit Decreasing (MBFD)
(POORANIAN et al., 2017)	Maximum Density Consolidation (MDC)
(DENG et al., 2016)	Hungarian Algorithm
(FARRIS et al., 2017)	Teoria dos Jogos
(AAZAM et al., 2016)	RR
(AAZAM et al., 2016)	NPS
(AAZAM et al., 2016)	Random Number Generator (RNG)
(RACHKIDI et al., 2016), (CARDELLINI et al., 2015), (VELASQUEZ et al., 2017),(LIU; LEE; ZHENG, 2016)	ILP
(SKARLAT et al., 2017a), (MENNES et al., 2016)	GA
(DO et al., 2015), (XU et al., 2017)	Alternating Direction Method of Multipliers (ADMM)
(GUPTA et al., 2016)	Edge-ward Placement
(HE et al., 2016)	MPSO-CO
(BAJPAI; CHOUDHURY; CHOUDHURY, 2017)	Adaptive Algorithm
(WANG et al., 2017), (TANEJA; DAVY, 2017)	Heurística Gulosa
(WANG et al., 2017)	Vineyard Algorithms
(ZENG et al., 2016)	MINLP
(SKARLAT et al., 2017a)	First-Fit
(URGAONKAR et al., 2015)	Lyapunov Optimization

3.2.2 Quais as ferramentas de simulação são mais utilizadas (Q2)?

Desenvolver um cenário real para avaliar o desempenho de políticas baseadas na *fog computing* é em muitos casos inviável financeiramente e de difícil escalabilidade. Assim, pesquisadores precisam se apoiar em ferramentas eficientes para a simulação de um ambiente *fog computing*. Nesta direção, a resposta para esta questão é ilustrada na figura 4, que apresenta um gráfico de frequência, representando as ferramenta utilizadas para validação dos experimentos gerenciamento de recursos em *fog computing*.

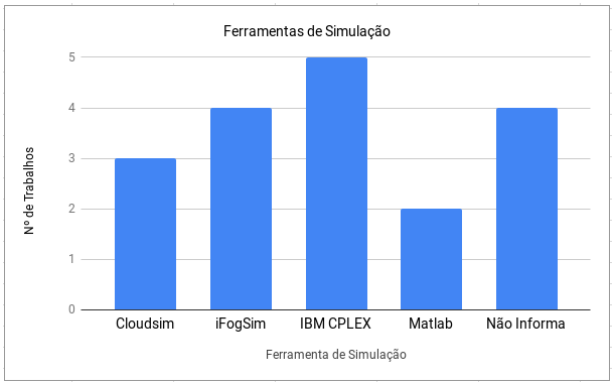


Figura 4 – Ferramentas de Simulação

Conforme resultado apresentado na tabela 4, a ferramenta IBM CPLEX (OPTIMIZER, 2016), é a mais utilizada nestas pesquisas. Esta informação reforça os resultados apresentados na questão 1, onde apresenta que as técnicas mais utilizadas neste tipo de problema é a programação linear. Outro aspecto importante a se observar é que, algumas ferramentas adotadas são extensões do CloudSim (BELOGLAZOV et al., 2016). A exemplo podemos citar, iFogsim (GUPTA et al., 2016).

Tabela 4 – Ferramentas de Simulação

Trabalhos	Ferramenta de Simulação
(AAZAM; HUH, 2015), (AGARWAL; YADAV; YADAV, 2016), (VERMA et al., 2016)	Cloudsim
(RACHKIDI et al., 2016), (BAJPAI; CHOUDHURY; CHOUDHURY, 2017), (WANG et al., 2017), (ZENG et al., 2016)	iFogSim
(POORANIAN et al., 2017),(SKARLAT et al., 2017a), (TANEJA; DAVY, 2017), (GUPTA et al., 2016), (MAHMUD; KOCH; BUYYA, 2018)	IBM CPLEX
(DENG et al., 2016), (FARRIS et al., 2017)	Matlab
(HE et al., 2017), (XU et al., 2017), (MENNES et al., 2016), (URGAONKAR et al., 2015)	Não Informa

3.2.3 Quais as plataformas de hardware são mais utilizadas (Q3)?

A resposta para esta pergunta é ilustrada na figura 5, que apresenta um gráfico de pizza representando os tipos de plataforma utilizadas em experimentos e na implementação da *fog computing*.

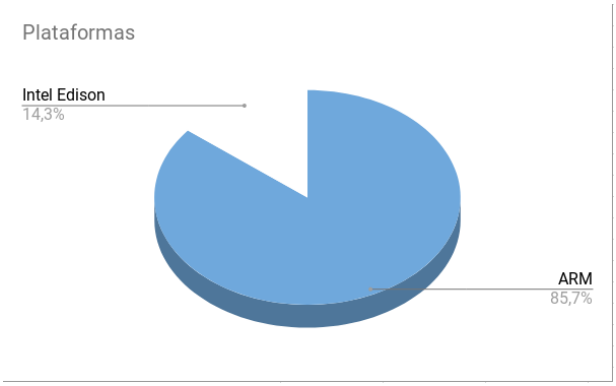


Figura 5 – Plataformas

O quadro 5 apresenta os artigos que utilizaram cada uma das plataformas apresentadas no gráfico da figura 5. A partir dos resultados apresentados, pode-se observar que os pesquisadores trabalham, em sua maioria, com a plataforma ARM. Um dos principais motivos para que isto ocorra pode estar relacionado ao custo e facilidade de obtenção destes dispositivos, grande oferta de dispositivos embarcados no mercado, também denominados SBC (*Single-Board Computers*). Como exemplo, é possível citar Raspberry Pi Zero, Raspberry 2 Model B, Raspberry Pi 3 Model B, entre outros. É digno de nota observar que apenas um dos trabalhos analisados utilizou a plataforma Intel Edison.

Tabela 5 – Plataformas

Autores	Plataformas
(RENNER; MELDAU; KLIEM, 2016), (CHARALAMPIDIS; TRAGOS; FRAGKIADAKIS, 2017), (DSOUZA; AHN; TAGUINOD, 2014), (HOQUE et al., 2017), (MORABITO et al., 2017), (PAHL; LEE, 2015)	ARM
(SAMANIEGO; DETERS, 2016)	Intel Edison

3.2.4 Quais são as métricas utilizadas na avaliação das abordagens (Q4)?

Métricas são indicadores de desempenho utilizados para transformar em números questões como desempenho, qualidade de serviço, entre outros aspectos . Quando bem utilizadas, as métricas ajudam a melhorar a qualidade dos serviços e servem como indicadores de prevenção de problemas no ambiente.

Neste trabalho, as métricas estão relacionadas ao tipo de recursos gerenciado na *fog computing*. Assim, os trabalhos de pesquisas relacionados são analisados com base no cumprimento de uma seleção de critérios importantes. A fim de agrupar e comparar adequadamente os trabalhos de pesquisas, os critérios incluem o tipo recurso do ambiente gerenciado, por exemplo, recursos computacionais, comunicação, energia e custo, bem como tópicos específicos abordados pelos autores. A figura 6, ilustra os resultados da quarta questão.

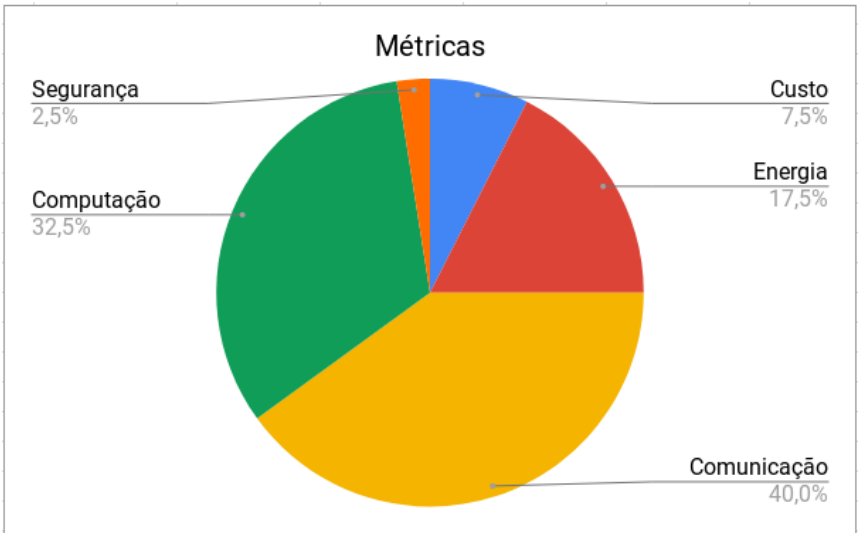


Figura 6 – Métricas

Comunicação: As métricas relacionadas à questões de comunicação no ambiente *fog computing* incluem aspectos como, latência e largura de banda. Em Renner, Meldau e Kliem (2016), os autores propuseram e avaliaram um esquema de alocação de recursos baseado no uso de contêineres, com o objetivo de reduzir o tráfego de dados gerado na rede por dispositivos inteligentes.

Computação: Esta categoria agrupa os trabalhos que tem por objetivo otimizar aspectos como a utilização de recursos computacionais tais como CPU, Memória, etc. No trabalho Rachkidi et al. (2016), os autores apresentaram um modelo de provisionamento de serviços IoT, formulado como um problema inteiro misto com o objetivo de otimizar a utilização de recursos em nós físicos e o consumo de largura de banda na rede.

Energia: tais como, consumo de energia e a sua relação entre consumo de desempenho. Em Suto et al. (2015), os autores propuseram um esquema de operação de um sistema de computação sem fio (WCS), com o objetivo de controlar o tempo de desligamento das antenas e a conectividade de rede para reduzir o consumo de energia enquanto satisfaz níveis aceitáveis de SLA.

Custo: Em certos casos, os fatores relacionados aos custos envolvem tanto os provedores de serviços quanto os usuários, tornando-se muito influentes no provisionamento de serviços na *Fog*. Isto inclui, adotar estratégias para reduzir o custo com a utilização de serviços providos

pela nuvem. Em Aazam e Huh (2015), os autores apresentam um modelo de gerenciamento de recursos orientado a serviços. A abordagem considera características do usuário para estimar valores para o provimento de serviços.

Segurança: Outro aspecto importante a ser considerado é o da segurança, uma vez que a *fog computing* é composta por uma rede de dispositivos heterogêneos, novos desafios precisam ser superados. No trabalho de Dsouza, Ahn e Taguinod (2014), os autores apresentam uma política de gerenciamento de recursos com suporte a colaboração segura e interoperabilidade entre diferentes níveis de QoS, baseado em um modelo de arquitetura *fog computing*.

A tabela 6 representa os critérios selecionados para a classificação dos trabalhos relacionados. É digno de nota observar que, embora existem esforços de otimização diferentes tipos de recursos, os resultados evidenciam que a maioria dos trabalhos analisados concentram-se em otimizar dois principais aspectos, comunicação e recursos computacionais, representando um total de 32,5% e 40% respectivamente do total de trabalhos analisados.

Tabela 6 – Métricas

Trabalho	Métricas
(AAZAM; HUH, 2015), (AAZAM et al., 2016), (MAHMUD; KOTAGIRI; BUYYA, 2018)	Custo
(SUTO et al., 2015), (CHARALAMPIDIS; TRAGOS; FRAGKIADAKIS, 2017), (POORANIAN et al., 2017), (DENG et al., 2016), (GUPTA et al., 2016), (MORABITO et al., 2017), (MAHMUD; KOTAGIRI; BUYYA, 2018)	Energia
(SUTO et al., 2015), (CARDELLINI et al., 2015), (CHARALAMPIDIS; TRAGOS; FRAGKIADAKIS, 2017), (SKARLAT et al., 2017a), (TANEJA; DAVY, 2017), (DENG et al., 2016), (DENG et al., 2016), (GUPTA et al., 2016), (HE et al., 2016), (XU et al., 2017), (BAJPAI; CHOUDHURY; CHOUDHURY, 2017), (HOQUE et al., 2017), (MORABITO et al., 2017), (URGAONKAR et al., 2015), (CARDELLINI et al., 2017), (MAHMUD; KOTAGIRI; BUYYA, 2018)	Comunicação
(RACHKIDI et al., 2016), (VERMA et al., 2016), (SKARLAT et al., 2017a), (TANEJA; DAVY, 2017), (SAMANIEGO; DETERS, 2016) (ZENG et al., 2016) (GUPTA et al., 2016), (BAJPAI; CHOUDHURY; CHOUDHURY, 2017), (MENNES et al., 2016), (MORABITO et al., 2017), (WANG et al., 2017), (ZENG et al., 2016), (MAHMUD; KOTAGIRI; BUYYA, 2018)	Computação
(DENG et al., 2016)	Segurança

3.2.5 Quais os domínios de aplicações podem se beneficiar com a implantação da *fog computing* (Q5) ?

A resposta para esta pergunta é apresentada na figura 7. Os domínios de aplicação são descritos de acordo com a área de atuação exploradas nos trabalho selecionados. A figura 7,

representada por um gráfico de barra, ilustra os resultados obtidos.

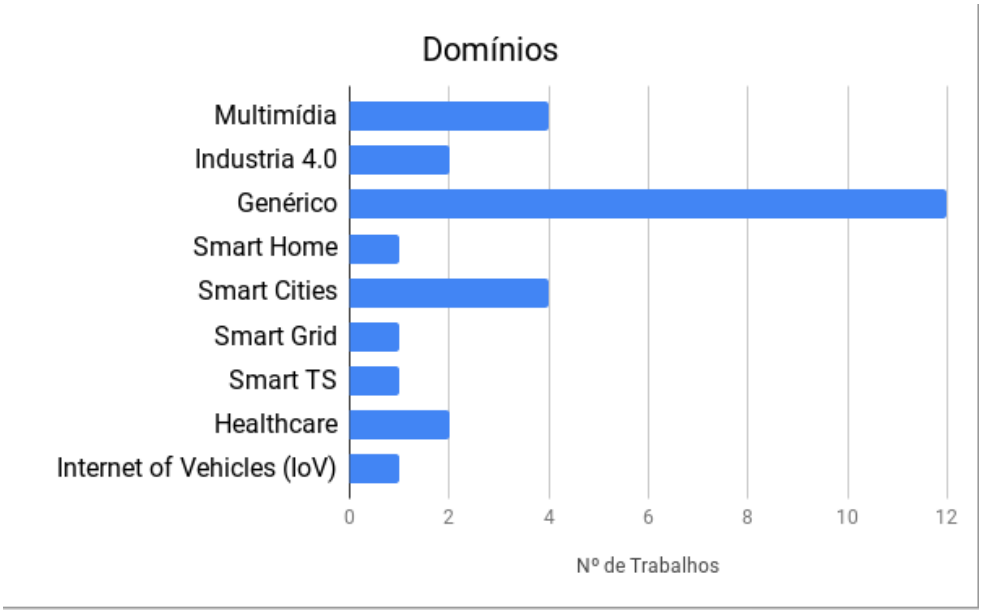


Figura 7 – Domínios de Aplicações

Multimídia: As aplicações multimídias podem ser consideradas como programas e sistemas em que a comunicação entre homem e o computador se dá através de múltiplos meios de representação da informação, como som e imagem animada. No âmbito da IoT, estas aplicações podem ser representadas como por exemplo, com a interação entre câmeras IP e um sistema de vigilância. Gupta et al. (2016) apresentaram uma estratégia para diminuir a latência de informações entre o sistema de detecção movimento e o ambiente de monitoramento.

***Smart Cities*:** O conceito de *smart cities* engloba a forma com que serviços essenciais tais como, energia, água, transporte, etc, são geridos em conjunto para proporcionar um ambiente limpo, econômico e seguro para o convívio das pessoas (GEISLER, 2013). Informações oportunas de logística são coletadas e fornecidas ao público por todos os meios disponíveis ao usuário, por exemplo, por meio de redes de mídia social.

Smart Grid: Este conceito define uma rede elétrica flexível, resiliente, performática segura e permitindo a exploração de recursos de rede em tempo real, de modo a otimizar a produtividade. A *fog computing* pode beneficiar esta área através do intermédio da implantação de milhares de dispositivos de controle, sensores e medidores inteligentes.

Smart Home: O conceito de *smart home* pode ser descrito como um conjunto de dispositivos e sensores conectados à ambientes domésticos. Além disso, para algumas tarefas, como por exemplo, análise de vídeo em tempo real, o ambiente necessita de recursos computacionais além do que é encontrado na maioria dos dispositivos IoT.

***Smart Transportation*:** Neste conceito sensores são embutidos na infraestrutura para monitorar informações de trânsito em tempo real e otimizar caminhos. Outras questões também

podem ser analisadas, tais como, fadiga estrutural, monitoramento de acidentes para gerenciamento de incidentes e coordenação de respostas de emergência.

Genérico: Diversos trabalhos apresentam soluções para o gerenciamento de recursos na *Fog*, contudo, não estão associados diretamente a nenhum domínio em particular. Para esta situação, agrupamos estes trabalhos em um domínio genérico, uma vez que, o método proposto pode beneficiar deferentes domínios de aplicações. Entre eles, podemos citar os trabalhos de Cardellini et al. (2015), Rachkidi et al. (2016) e Morabito et al. (2017).

***Healthcare*:** Aplicações *healthcare* são destinadas a dar apoio a questões ligadas à saúde e bem estar das pessoas. Atividades, como triagem, monitoramento de pacientes, monitoramento de pessoal são algumas das que podem colaborar com informações preditivas, para ajudar a tomada de decisão de profissionais médicos ou decisões políticas em cenários de pandemia.

Internet of Vehicles (IoV): Em aplicações IoV, a *fog computing* pode ser integrada em redes veiculares, permitindo a criação autônoma de uma comunicação sem fio entre veículos para troca de dados e aumento de recursos.

Como pode-se analisar, a grande maioria dos trabalhos não apresentam um domínio de aplicação específico. No quadro 7 são apresentados os trabalhos analisados de acordo com a área de domínio de cada aplicação.

Tabela 7 – Domínios de Aplicação

Trabalhos	Domínio
(AAZAM; HUH, 2015), (AGARWAL; YADAV; YADAV, 2016), (DO et al., 2015), (GUPTA et al., 2016), (WANG et al., 2017)	Multimídia
(SUTO et al., 2015), (SKARLAT et al., 2017a)	Industria 4.0
(CARDELLINI et al., 2015), (RACHKIDI et al., 2016), (VERMA et al., 2016), (TANEJA; DAVY, 2017), (BAJPAI; CHOUDHURY; CHOUDHURY, 2017), (MENNES et al., 2016), (HOQUE et al., 2017), (MORABITO et al., 2017), (URGAONKAR et al., 2015), (SAMANIEGO; DETERS, 2016), (XU et al., 2017), (FARRIS et al., 2017)	Genérico
(RENNER; MELDAU; KLIEM, 2016)	Smart Home
(CHARALAMPIDIS; TRAGOS; FRAGKIADAKIS, 2017), (DENG et al., 2016), (VELASQUEZ et al., 2017), (MASIP-BRUIN et al., 2018)	Smart Cities
(POORANIAN et al., 2017)	Smart Grid
(DSOUZA; AHN; TAGUINOD, 2014)	Smart Transportation System
(GUPTA et al., 2016), (MAHMUD; KOTAGIRI; BUYYA, 2018)	Healthcare
(HE et al., 2017)	Internet of Vehicles (IoV)

3.3 Considerações Finais do Capítulo

O processo seguido para realização do mapeamento consistiu da criação de questões de pesquisa que nortearam o desenvolvimento da busca e seleção de artigos para análise mais aprofundada. De modo que foram encontrados 1.421 artigos em cinco bases de dados distintas (Scopus, IEEE Xplorer, ScienceDirect, Springer e ACM), sendo que destes, ao serem aplicados os critérios de seleção, reduziram-se a 33 artigos considerados relevantes para o objetivo proposto.

A partir da análise dos estudos primários pode-se encontrar as diferentes estratégias de gerenciamento de recursos a serem desenvolvidos na área da *fog computing*. Descobriu-se ainda que a maioria dos trabalhos selecionados utilizam algoritmos bastante diversificados, tendo destaque para os algoritmos genético e ILP, que já são bem consolidados em trabalhos relacionados a ambientes de nuvem (Q1).

Além disso, embora os resultados das ferramentas de simulação não apresentem uma ferramenta com maior destaque. iFogsim é uma extensão do Cloudsim, o que nos permite concluir a partir destas informações, que Cloudsim é a ferramenta mais utilizada para simulação de ambientes *fog computing* (Q2). Por outro lado, no que se diz respeito ao tipo de plataforma utilizada nos trabalhos, concluímos que os pesquisadores trabalham, em sua maioria, com plataforma ARM (Q3). Não obstante, a maioria das abordagens analisadas deram ênfase a otimização de aspectos de comunicação e computação (Q4). Os trabalhos analisados trazem benefícios para diversas áreas de domínio de aplicação *fog computing*, contudo, a maioria dos trabalhos optaram desenvolver as estratégias sem associá-las a um determinado domínio (Q5).

Em um cenário de *fog computing*, o provisionamento de recursos é ainda mais complicado em comparação a um ambiente de nuvem, uma vez que dispositivos heterogêneos distribuídos precisam ser provisionados de acordo com latência e tempos de execução de tarefas. Além disso, a *fog computing* é composto por uma hierarquia dinâmica. Assim, a hierarquia de dispositivos pode mudar durante o tempo de execução e, portanto, não está disponível no início do serviço. Neste contexto, o gerenciamento de recursos engloba atividades que descrevem todas as características dos recursos. Assim, o provisionamento de recursos *fog computing* pode ser estendido para o desenvolvimento de um ambiente autonômico.

Nesta direção, diversos trabalhos de pesquisa apresentam soluções para lidar com questões relacionadas ao IoT na borda da rede, fornecendo provisionamento e escalonamento de recursos e serviços de forma eficiente.

Os trabalhos de pesquisa relacionados são analisados com base no cumprimento de uma seleção de critérios importantes. A fim de agrupar e comparar adequadamente o trabalho de pesquisa, os critérios incluem a abordagem proposta em relação à alocação de recursos e o tipo de recurso gerenciado, bem como tópicos específicos abordados pelos autores. A tabela 8 representa os critérios selecionados para a classificação dos trabalhos relacionados.

Tabela 8 – Visão geral de alguns aspectos da revisão bibliográfica

Autor	Tipo do recurso			Abordagens de alocação de Recursos		
	Computacional	**Comunicação**	**Energia**	**Provisionamento**	**Escalonamento**	**Autonômico**
Skarlat et al (SKARLAT et al., 2017b)	x	x		x		
Cardellini et al. (CARDELLINI et al., 2015)		x		x		
Taneja et al. (TANEJA; DAVY, 2017)	x	x	x	x		
Aazam et al. (AAZAM; HUH, 2015)				x		
Xu et al. (XU et al., 2014)		x		x		
Bajpai et al. (BAJPAI; CHOUDHURY; CHOUDHURY, 2017)	x	x		x		
Mennes et al. (MENNES et al., 2016)		x		x		
Wang et al. (WANG et al., 2017)		x		x		
Velasques et al. (VELASQUEZ et al., 2017)	x	x		x	x	
Tarneberg et al. (TÄRNEBERG et al., 2017)	x	x			x	
Urgaonkar et al. (URGAONKAR et al., 2015)		x		x	x	
Rodrigues et al. (RODRIGUES et al., 2017)		x	x		x	
Yousaf et al. (YOUSAF; TALEB, 2016)	x	x			x	
Este trabalho	x	x	x	x	x	x

É crescente o surgimento de soluções da IoT que buscam resolver problemas de alocação de recursos. Além disso uma promissora direção de pesquisa parte do princípio de explorar interações diretas entre coisas com o objetivo de monitorar e controlar a si mesmos e o ambiente

circundante com a mínima intervenção humana. Nesta direção, abordagens autonômicas são amplamente exploradas na literatura em contextos de computação em nuvem como solução para problemas de gerenciamento de recursos , (SINGH; CHANA, 2016)(SINGH; CHANA; SINGH, 2017)(GUÉROUT; ALAYA, 2013). Contudo, ao melhor do nosso conhecimento, entendemos que abordagens autonômicas para o provisionamento de recursos de computação um nuvem não podem ser aplicadas diretamente em um cenário de *fog computing*, mas é possível extrair a ideia geral dessas abordagens de computação em nuvem e usá-las como um ponto de partida .

4

Trabalho Proposto

Este capítulo apresenta os aspectos de concepção da abordagem proposta, a qual constitui a contribuição central desta dissertação de mestrado. Neste sentido, são discutidas as premissas de concepção da arquitetura autonômica e as funcionalidades desenvolvidas.

4.1 Modelo do Sistema

A arquitetura de referência para o nosso modelo é descrita em (BONOMI et al., 2014), representada por três camadas distribuídas entre nuvem, *fog* e IoT ilustradas na figura 8.

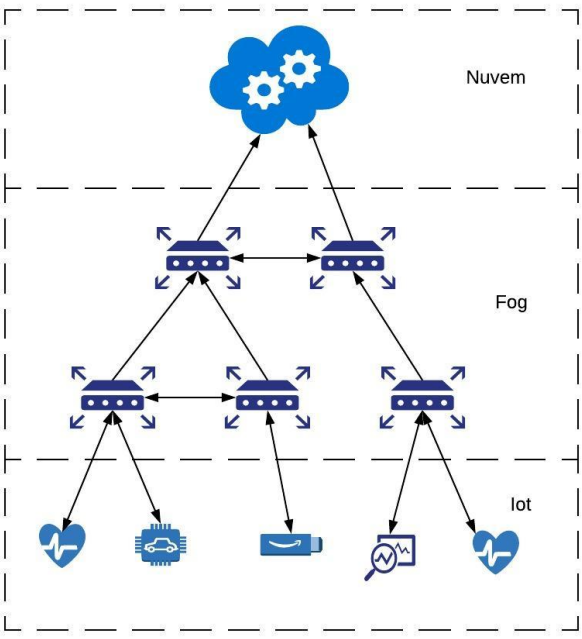


Figura 8 – Arquitetura *Fog Computing*

Nuvem: A nuvem localiza-se na camada mais superior e interage como suporte para execução de aplicações não suportadas na fog.

Fog: A camada Fog compreende os dispositivos estacionários de rede localizados entre a nuvem e os dispositivos IoT, tais como roteadores, *access point* e *edges*. Na concepção da arquitetura proposta, consideramos a possibilidade do gerenciamento funcionar independente da nuvem computacional e ela somente será utilizada caso os nós são tenham recursos suficientes. Por esta razão, classificamos os dispositivos em dois tipos, *Fog Control Nodes* (FCN) e *Fog Nodes* (FN). FCN são um tipo específico de *fog node*, geralmente *gateways* IoT, que possuem maior poder computacional e maior largura de banda disponível. Este tipo de nó suporta os quatro módulos da arquitetura que será apresentada com mais detalhes na seção 4.3 e dão suporte a orquestração dos demais nós. Os *fog nodes* são limitados ao uso dos módulos de monitoramento e possuem a função de também hospedar os módulos das aplicações.

IoT: Esta camada é composta por sensores e atuadores que formam o terceiro componente vital da IoT. Estes são frequentemente conectados aos dispositivos computacionais periféricos por *links* físicos, redes sem fio *ad hoc* ou até mesmo a bordo do dispositivo. Eles formam a origem dos fluxos de dados distribuídos e que são intrínsecos às implantações de aplicações IoT.

Enquanto os dispositivos IoT estão sob o nível inferior, a camada *fog* e nuvem compreendem o segundo e o terceiro nível, respectivamente, formando juntos uma arquitetura de três camadas. Nesta arquitetura, consideramos como um nó da *fog*, qualquer elemento capaz de hospedar ao menos uma módulo de aplicação. Assim, a capacidade computacional de um nó na *fog* pode ser descrita como um conjunto de constantes finitas que representam os atributos dos recursos de CPU, memória, disco e largura de banda.

A representação do modelo de sistema da arquitetura é apresentado na subseção seguinte e é dividido em dois tópicos descritos como, modelo de aplicação e modelo de recursos. O primeiro, diz respeito aos atributos definidos para cada aplicação e a sua notação matemática. Por outro lado, o modelo de recursos representa os atributos de cada recurso, sua notação, bem como, as limitações para que cada aplicação seja alocada em cada nó da *fog*. Estes modelos servem como base para o desenvolvimento dos algoritmos de provisionamento, uma vez que seus atributos se relacionam para o provimentos de métodos de tomada de decisão e restrições de alocação.

4.1.1 Modelo da Aplicação

Uma aplicação IoT consiste em vários serviços, que são executados em dispositivos IoT virtualizados, ou seja, dispositivos *fog*, e interagem uns com os outros para fornecer funcionalidades em comum. As instâncias de *software* reais que executam esses pedidos de tarefas são chamadas de serviços. Possíveis exemplos de serviços incluem processamento de fluxo de dados, aplicações multimídia, ou armazenamento de dados distribuídos (SKARLAT et al., 2017a). O

ambiente de computação distribuída exige componentes distribuídos, o que proporciona melhores resultados com aplicações compostas por vários componentes.

A distribuição dos módulos entre os nós considerando os níveis de QoS pode ser considerado um problema NP-completo (HUANG; GANAPATHY; WOLF, 2009). Geralmente, este tipo de problema é resolvido com força bruta, contudo, utilizar estratégias de força bruta podem levar muito tempo para alcançar uma solução ideal, o que é contrastante com a nossa abordagem. Neste trabalho, a aplicação será representada por contêineres, representando os módulos de aplicação, onde cada módulo possui um requisito representado inicialmente como um conjunto de quatro atributos finitos descritos como, CPU, memória RAM, disco e largura de banda. Portanto, se a_{i} representa um módulo i na aplicação, o requisito do referido módulo é representado como na equação 1:
Seja A o conjunto de todos os módulos de uma aplicação.

$$A = ai \quad (2)$$

O tipo de aplicação compatível com esta arquitetura é baseada no modelo *Distributed Data Flow (DDF)* (GIANG et al., 2015)e seu modelo é representado por um grafo direto acíclico (DAG), onde os módulos são representados pelos vértices e a dependência de dados entre os módulos da aplicação são representados pelas arestas. A notação matemática de um DAG é representada pela tupla T na equação (3). Onde,

$$T =< A, E > \quad (3)$$

4.1.2 Modelo dos Recursos

A capacidade computacional de um nó (equação 4) de rede é definida por um conjunto geral de restrições finitas e pode ser representada como um conjunto de quatro atributos básicos, ou seja, CPU, RAM e Disco e Largura de Banda. Os recursos são modelados de acordo com as suas capacidades, requisitos e restrições definidos a seguir.

$$Cap(n_{\mathrm{i}}) =< CPU_{\mathrm{i}}, RAM_{\mathrm{i}}, Disco_{\mathrm{i}}, LarguradeBanda_{\mathrm{i}} > \quad (4)$$

O conjunto de todos os elementos computacionais contidos na infraestrutura são representados pelo conjunto N, onde,

$$N = n_{\mathrm{i}} \quad (5)$$

N pode ser dividido em dois subconjuntos, F_c e F_n onde,

F_c = Conjunto de todos os nós de controle na camada *fog* (6)

F_n = Conjunto de todos os nós associados a um nós de controle na camada *fog* (7)

$$F_c \cup F_n = N (8)$$

$$F_c \cap F_n = \emptyset (9)$$

O mapeamento das aplicações no recursos é definida pela função M, onde,

$$M : A \rightarrow N \ (10)$$

Que indica o nó da rede no qual o módulo da aplicação é alocado, de forma que ele atenda ao seguinte:

$$\forall (a_i, n_i) \in M \ \Big| \ \Rightarrow (Req(a_i) \leq (Cap(n_i)), \forall (a_i) \in A, \forall (n_i) \in N \quad (11)$$

4.2 Visão Geral da Arquitetura

A arquitetura proposta nesta dissertação baseia-se no modelo autonômico da IBM (COMPUTING et al., 2006), que considera quatro etapas do sistema autônomo: Monitor, Analise, Planejamento e Execução. Além disso, estes módulos compartilham uma base de conhecimento para armazenar e coletar detalhes dos recursos gerenciados, políticas, e sintomas, que e constituem o laço de controle MAPE-K (COMPUTING et al., 2006). A decisão sobre alocar, re-alocar ou desalocar recursos deve ser tomada com base em indicadores reais de desempenho, quantidade de recursos computacionais em uso pelo sistema e requisitos de QoS. A figura 9 ilustra os principais módulos do laço de controle e seu fluxo de trabalho.

4.2.1 Monitor

O monitoramento de recursos é uma importante tarefa, uma vez que a alocação de recursos depende diretamente da disponibilidade de recursos de cada um dos nós do ambiente fog. Para permitir o monitoramento, os nós utilizam uma aplicação de monitoramento executada diretamente no sistema do *host*. Sua principal função é coletar informações a respeito da utilização de recursos de cada nó, tais como, uso de CPU, memória, espaço em disco e rede e os envia ao nó de controle, que por sua vez recebe informações sobre a utilização e disponibilidade de recursos dos nós adjacentes e as fornece de forma centralizada para o módulo de análise.

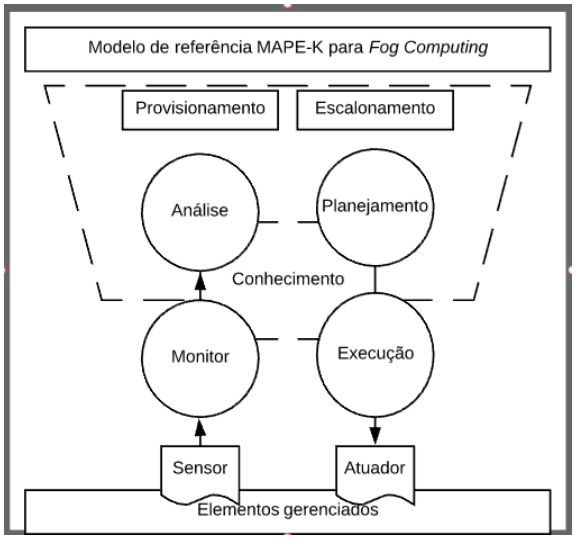


Figura 9 – Visão geral da arquitetura

A coleta de dados é periódica, podendo ser definida pela administrador e utiliza como base o padrão de engenharia de *software* Publisher-Subscriber (ZU; BAI; YAO, 2016), onde o *software* define uma dependência de um-para-muitos entre objetos de modo que quando um nó muda o estado, todos seus dependentes são notificados e atualizados automaticamente. O sistema monitora periodicamente os recursos dos sistemas e salva a utilização resultante de arquivos JSON para processamento adicional.

4.2.2 Análise e Planejamento

Os módulos de análise e planejamento contém diferentes algoritmos que procuram um mapeamento adequado para a distribuição das tarefas de acordo com a disponibilidade dos nós ou através de alertas gerados por sintomas gerados no ambiente. No contexto autonômico, o termo sintoma está relacionado a acontecimentos que desencadeiam um evento. Os algoritmos e estrategias relacionados aos módulos de análise e planejamento são apresentados em detalhes na seção 4.3.

4.2.3 Execução

Este módulo fornece os mecanismos que controlam a execução de um plano com consideração para atualizações dinâmicas. Ele recebe como entrada uma sequencia de ações mapeadas pelos módulos de planejamento e as executa usando atuadores de entidades gerenciadas. O executor é capaz de executar *scripts* de ação usando diferentes tipos de tecnologia, como por exemplo, *Shell Script*, *Python* ou APIs.

4.3 Estratégias de Otimização

Esta seção descreve estratégias de otimização como reação a eventos autonômicos no ambiente *fog computing*. Neste contexto, dois tipos de eventos são definidos no escopo deste trabalho. O primeiro diz respeito à solicitação inicial de uma carga de trabalho que precisa ser distribuída entre os nós da rede para atender a requisições de sensores IoT localizados na borda da rede. O segundo caso acontece quando o módulo da aplicação precisa ser migrado para outro nó para atender à algum objetivo definido de QoS, como por exemplo, quando um nó está subutilizado em relação a um limite de carga, ex,. 10%. Em outras palavras, o primeiro objetivo diz respeito onde os serviços serão alocados e em um segundo momento, "quando" migrar o serviço considerando uma melhor execução.

4.3.1 Alocação de Serviços

O objetivo desta técnica é determinar o mapeamento de aplicações IoT entre os nós da rede de acordo com objetivos pré estabelecidos para definir onde a tarefa pode ser executada e quais recursos serão alocados para a melhor execução das tarefas. A figura 10 ilustra um exemplo de alocação de serviços entre os nós do ambiente *fog*. Este exemplo é composto por uma aplicação distribuída entre três módulos (a1, a2 e a3), onde o sensor conectado ao nó Edge-1.1 envia os dados para o módulo a1, que envia os dados para os módulos a2 e a3, que por fim os envia para um atuador conectado ao nó Edge-1.2. Uma estratégia que priorize a locação dos módulos entre os nós com menor latência e os objetivos de otimização podem ser adaptados para atender à demanda e trazer benefícios a diversos tipos de aplicações.

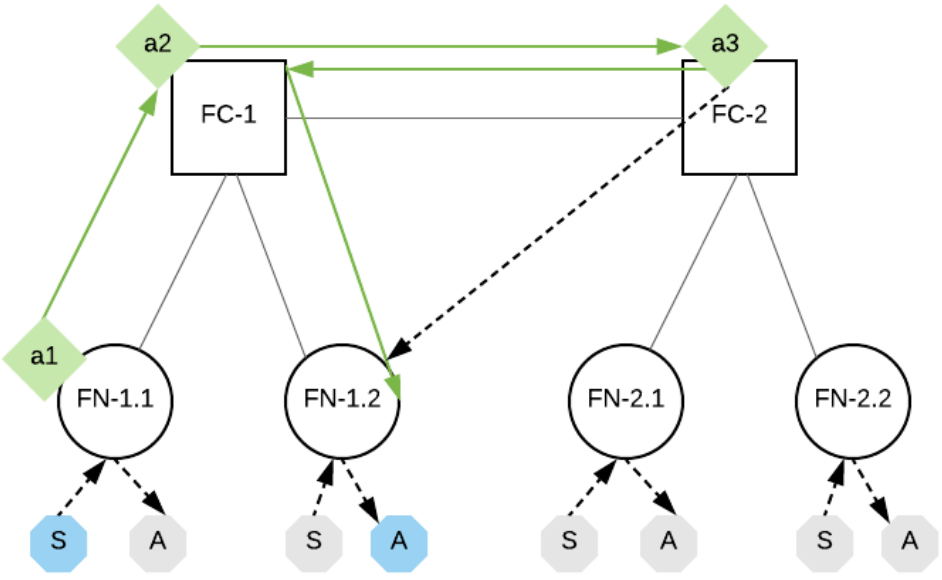


Figura 10 – Exemplo de alocação de serviços

A seguir descrevemos duas estratégias com diferentes tipos de algoritmos, o primeiro (Fist-Fit) que utiliza estratégia gulosa e o segundo (Algoritmo Genético), com uma estratégia evolucionária.

- **First-Fit**: é um algoritmo de heurística gulosa que fornece uma solução rápida para o mapeamento porém, não necessariamente eficiente, onde coloca-se cada módulo de aplicação para a primeira posição que puder acomodá-la na lista de nós disponíveis. O algoritmo 1 recebe como entrada duas listas, a primeira contém todos os nós disponíveis no ambiente *fog* e na nuvem e a segunda, um conjunto de tarefas a serem executadas. Em seguida, as listas são ordenadas de forma ascendente de acordo com a capacidade disponível e os requisitos da aplicação respectivamente. A lista de nós é percorrida a partir do nó de menor capacidade com o objetivo de otimizar a utilização dos recursos do ambiente. O laço interno percorre a lista de tarefas e chama um método que verifica se o nó é capaz de comportar a tarefa de acordo com os recursos disponíveis e em caso positivo, inclui o par na lista de mapeamento. Por fim, o algoritmo retorna a lista com o mapeamento.

Algorithm 1: Pseudocódigo First Fit

```
Data: Lista de nós N, Lista de Tarefas T
Result: Mapeamento das aplicações entre os nós
mapeamento = [];
ordene(n);
ordene(t);
inicialização;
for n - 1 do
    for t - 1 do
        utilizacao <– getUtilizacao(n);
        if verificaRestricoes(utilizacao) then
            mapeamento.add(n, t);
        end
    end
end
retorna (mapeamento);
```

- **Algoritmo Genético**: O algoritmo genético (GA) é inspirado na Teoria da Evolução das Espécies proposta por Charles Darwin, baseando-se nos princípios de sobrevivência dos mais aptos à reprodução (LUKE, 2017). Segundo a Teoria de Darwin, características individuais são transmitidas de pais para filhos e os indivíduos melhor adaptados ao ambiente têm maior chance de sobreviver e, por consequência, de passar suas características a um número maior de descendentes. Algoritmos genéticos partem de um conjunto de possíveis soluções geradas aleatoriamente (população inicial) compostas por cromossomos (ou indivíduos) que consistem em um conjunto de genes com o mesmo comprimento.

Esses algoritmos incluem uma função de *fitness* que atribui um valor a cada indivíduo com base na proximidade da solução ótima. Para se aproximar das soluções ótimas, diversos operadores genéticos são aplicados à população, tais como, operadores de mutação, operadores de seleção e *crossover*. O algoritmo 2 representa o pseudocódigo descrito nesse trabalho onde, t representa o tempo atual, d representa o tempo determinado para finalizar a execução do algoritmo e P representa a população.

Algorithm 2: Pseudocódigo Algoritmo Genético

```
Data: Solucao candidata P
Result: Mapeamento das aplicações entre os nós
t = 0;
iniciaPopulação (P, t);
avaliação (P, t);
for n - 1 do
    t = t +1;
    seleçãoPais (P,t);
    cruzamento (P, t);
    mutação (P, t);
    avaliação (P, t);
    sobrevivem (P, t)
end
retorna (mapeamento);
```

Representação do cromossomo: A codificação do cromossomo ilustrado na figura 11), é um vetor que representa um plano de mapeamento de alocação para os serviços. O número total de serviços para uma determinada tarefa representa o comprimento do cromossomo. Cada gene no cromossomo representa um módulo de aplicação a ser alocado e um índice que referencia o nó que atenderá a demanda.

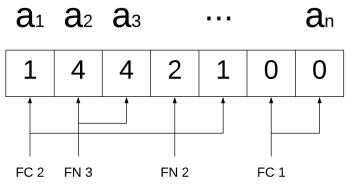


Figura 11 – Representação do Cormossomo

Função de *Fitness*: A função de *fitness* é usada para avaliar a qualidade da solução candidata, que é um índice importante para selecionar o melhor indivíduo. Normalmente, o valor da função de *fitness* pode ser usado diretamente para a medição para determinar a proximidade da solução ótima. A otimização pode ser vista como um problema de minimização ou maximização da alocação de recursos obtendo pontos de mínimo ou máximo para uma função qualquer. Para o propósito de minimizar a latência para o conjunto de nós de uma solução a função objetivo

pode ser formulada como dado na Equação (12).

$$f(x) = \sum_{k=1}^{N} N_{\mathrm{L}} \quad (12)$$

No entanto, é possível estender a outros objetivos de otimização, tais como, minimizar a utilização de recursos de dispositivos, ou o consumo de energia na *fog computnig*.

4.3.2 Migração de Serviços

Um ponto importante no processo de otimização da utilização de recursos em um ambiente *fog* é definir como os serviços serão migrados de modo a não comprometer o provimento dos mesmos. Os *data centers* convencionais residem a uma distância de vários saltos das fontes de dados de IoT, o que também pode aumentar a latência na propagação de dados. Assim, este módulo tem como recebe como solicitações de entrada o mapa de provisionamento gerado pelo módulo de análise e gera planos de ações como resposta e fornece os mecanismos que constroem ação necessária para atingir os objetivos determinado pelo administrador do ambiente. Esta camada é a parte principal do presente trabalho e por isso é apresentada com mais detalhes na seção 4.4.

4.4 Migração de Serviços e Alta Disponibilidade

A estratégia de migração inclui decidir quando a tarefa deverá ser migrada para outro *host* e propor técnicas para que a migração seja possível. As informações monitoradas no ambiente resultam em eventos autonômicos, onde o objetivo é encontrar o melhor *fog node* em termos de latência para executar a tarefa sem violar os limites de QoS. Na tabela 9, são apresentados valores de referência para limites de QoS para diferentes tipos de aplicação IoT. Estes valores representam as limitações em termo de latência, *jitter*, taxa de perda de pacotes e largura de banda para diferentes domínios.

Tabela 9 – Requisitos de QoS para Aplicações IoT. Adaptado de (SUÁREZ-ALBELA et al., 2017)

Domínios	**Latência (ms)**	**Jitter (ms)**	**Taxa de Perda de Pacotes**	**Largura de Banda (Mb/s)**
Indústria 4.0	5	0.5	10^{-4}	0.2
Big Data Streaming	100	10	10^{-2}	10
Smart City	10	3	10^{-3}	2

A seguir, a figura 12 (1) representa um estágio inicial de alocação onde os nós são distribuídos inicialmente entre o ambiente *fog* e a nuvem. Após a execução dos algoritmos de análise o módulo de planejamento define um novo mapeamento, figura 12 (2), neste exemplo o

objetivo é diminuir o número de saltos entre os nós e consequentemente, prover o serviço com uma menor latência. Em termos de consumo de energia, é possível obter economia migrando as tarefas para o menor número possível de nós, desligando os nós ociosos ou variando a frequência dos processadores de acordo com a carga de trabalho através de técnicas de DVFS (*Dynamic Voltage and Frequency Scaling*) (WU; CHANG; CHAN, 2014).

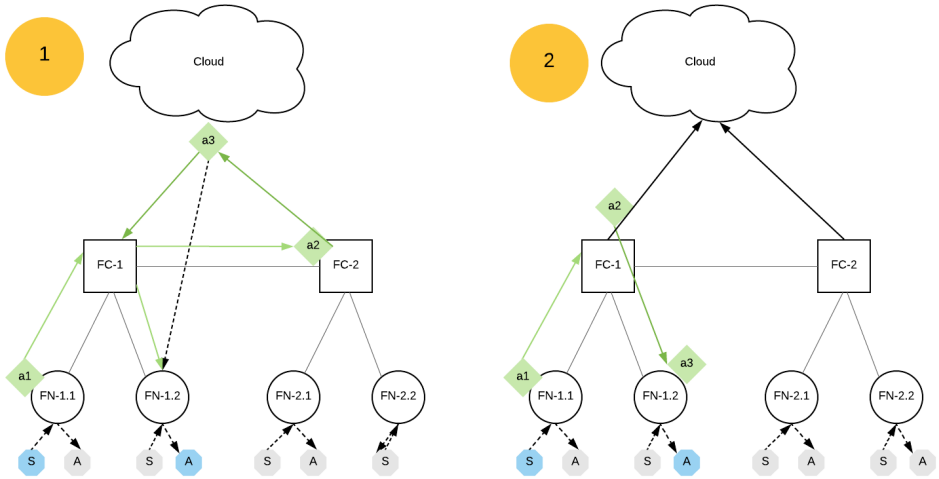


Figura 12 – Política de migração de serviços

Para realizar a tarefa de migração de tarefas entre nós na *fog computing*, é necessário um mecanismo que permita fazer uma imagem o estado atual do serviço para em seguida restaurá-lo no nó de destino. CRIU (*Checkpoint/Restore In Userspace*) é uma uma ferramenta que implementa funcionalidades de *checkpoint* e *restore* em ambientes Linux e possibilita a migração de contêineres em implementação de políticas de alocação de recursos para garantir a manutenção da qualidade de serviço. Além disso, provê alta disponibilidade através de uma sequências de passos conforme ilustrado na figura 13.

O fluxo do algoritmo de migração ilustrado na figura 13 representa a sequência de passos necessários para execução do processo de *live migration*, onde os módulo selecionados são movidos entre dois nós no ambiente *fog* (origem/destino), através de uma sequência de ações, enquanto o serviço continua disponível para os usuários. No primeiro passo, o nó de controle envia um comando para o nó de destino que fará o download da imagem do módulo através de um repositório de imagens localizado na nuvem. Em seguida, o nó de origem cria um *checkpoint* do módulo da aplicação. O *checkpoint* representa o estado atual da aplicação e será restaurado no nó de origem para dar continuidade à execução da aplicação a partir do momento exato em que foi feito o checkpoint. A sincronização entre os sistemas de arquivo é responsável por enviar os dados do *checkpoint* do nó de origem para o nó de destino, utilizando a ferramenta *Rsync*. Após a conclusão do *Rsync*, o módulo é criado no nó de destino e o *checkpoint* é restaurado. Por fim, o nó de controle envia um comando para finalizar e excluir o módulo de aplicação no nó de origem.

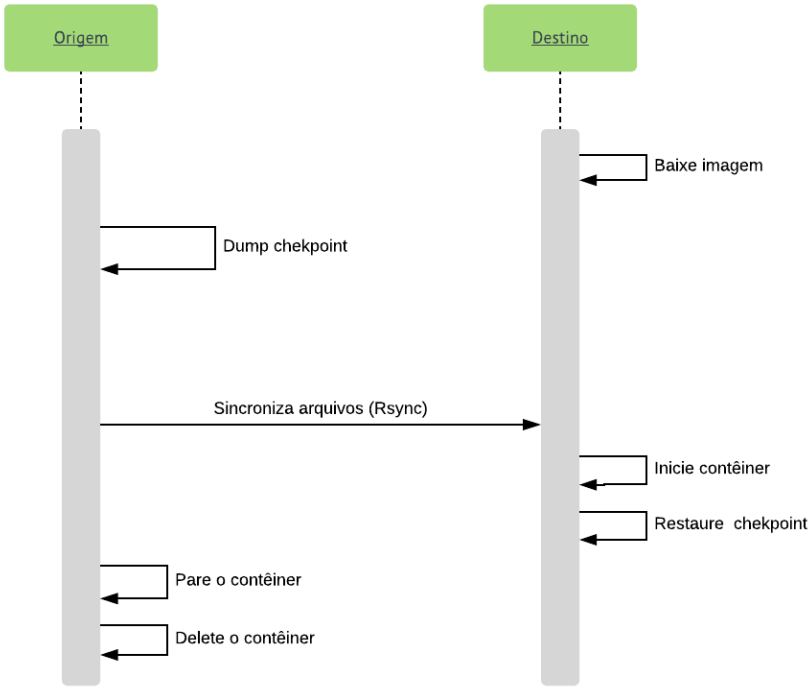


Figura 13 – Fluxo do algoritmo de migração

4.5 Experimentos e Avaliação

Neste capítulo, avaliamos as estratégias apresentadas na seção 4 e apresentamos os benefícios da *fog computing* em termos de otimização da qualidade de serviço para aplicações IoT com relação à da latência, e utilização de recursos.

4.5.1 Metodologia de Avaliação

Para realizar uma prototipação deve-se pressupor a existência de um modelo computacional idêntico ao ambiente real de produção. Porém, o número de equipamentos necessários para desenvolver um ambiente real pode inviabilizar financeiramente o desenvolvimento do trabalho e trazer complexidade quanto à configuração do cenário do experimento. Nesse intuito, ferramentas de simulação e emulação são úteis para a construção deste ambiente, uma vez que é possível aumentar o número e a complexidade de cenário.

4.5.1.1 VIoLET - A Large-scale Virtual Environment for Internet of Things

Neste trabalho utilizamos o VIoLET (*A Large-scale Virtual Environment for Internet of Things*) (BADIGER; BAHETI; SIMMHAN, 2018). VIoLET é um emulador desenvolvido com o uso de tecnologias de *software* livre e é capaz de emular os dispositivos *fog* utilizando contêineres Docker. Além disso, é possível no VIoLET é possível definir características de rede tais como, largura de banda e latência entre os dispositivos e recursos, como CPU e memória

RAM e disco. A escolha desta ferramenta deu-se pela flexibilidade da implementação e pela possibilidade de utilização de aplicações reais para validação dos experimentos. Utilizando como referência trabalhos correlatos e especificações técnicas de equipamentos disponíveis comercialmente, definimos as características de *hardware* para a validação do trabalho conforme apresentado na tabela 10.

Tabela 10 – Características de Hardware

Dispositivo	Tipo de Nó	Nº de Nós	Nº de núcleos	Memória	Coremark	Total Coremark
Raspberry Pi 2B	Edge	4	4	1GB	8,910	35,64
Raspberri Pi 3B	Edge	4	4	1GB	13,717	54.86
NVidia TX1	Fog	1	4	4GB	26,371	26,37
Softiron O. 3000	Fog	1	8	16GB	76,223	76,22
Total		**10**	**20**			193.09

Conforme apresentado na tabela 10, consideramos uma topologia física composta por 02 *Fog Control Nodes/Gateway* (NVidia TX1, Softiron Overdrive 3000 respectivamente), cada *gateway* está associado a 04 dispositivos de borda ou seja, 04 Raspberry pi 2B e 04 Raspberry pi 3B, que por sua vez estarão associados aos seus respectivos sensores e atuadores.

Os requisitos de CPU para a implementação do emulador ViOLET são definidos através do valor médio de iterações/Sec de cada dispositivo, obtido através da ferramenta Coremark. O valor total necessário para execução deste experimento é 193 coremarks, o que é equivalente a 02 máquinas virtuais Centos 7, com 16 núcleos cada, virtualizadas em um servidor Dell PowerEdge R530 com processador Intel® Xeon® E5-2620 v4 2.1GHz,20M Cache, 8.0GT/s QPI,Turbo,HT,8C/16T (85W) Max Mem 2133MHz, 32GB de memória e sistema Operacional Ubuntu 16.04. A comunicação entre os nós no ambiente virtual dá-se através do uso de *drivers overlay* ou seja, por uma abstração criada no docker para facilitar o gerenciamento da comunicação de dados entre contêineres e os nós externos ao ambiente. Além disso, é possível determinar características de rede específicas tais como latência, largura de banda e definir as características computacionais dos nós tais como CPU, memória e armazenamento. Implementamos um ambiente com uma rede pública, que conecta os *gateways* ente si, e duas redes privadas distintas formando subgrupos de trabalho entre os nós do ambiente *fog* (Fog-1 e Fog-2) e os dispositivos de borda associados a ele (*Edges*), tais como *Smartphones, tablets* e computadores de propósito geral. A figura 14 ilustra o ambiente virtual e as suas características de rede.

As características de rede esperada para as redes publicas e privadas configuradas no VIoLET são apresentadas na tabela 11.

Implementamos uma extensão no VIolET (BADIGER; BAHETI; SIMMHAN, 2018) para tornar possível a coleta de informações em tempo real durante a execução dos experimentos. O módulo de monitoramento é composto pelas ferramentas Grafana (GRAFANA,) e Prometheus (PROMETHEUS,), apresentadas nas subseções a seguir.

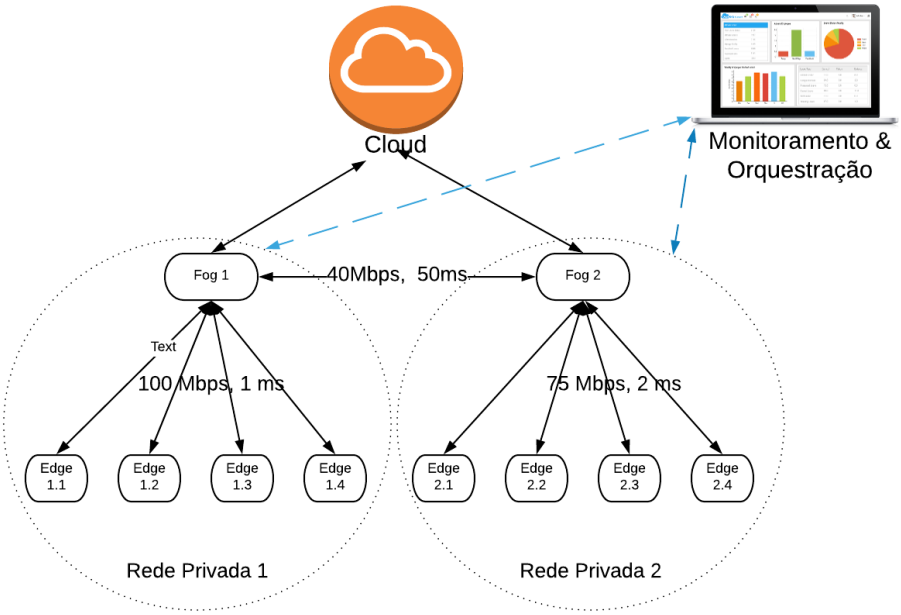


Figura 14 – Topologia de rede

Tabela 11 – Características de redes públicas e privadas esperadas

Rede	**Características**	
	Largura de banda (Mbps)	**Latência (ms)**
Rede Privada 1	100	1
Rede Privada 2	75	2
Rede Pública	40	50

4.5.1.2 Grafana

Grafana Grafana () é uma aplicação de visualização de dados que pode ser integrado a diferentes mecanismos para prover monitoramento em tempo real de aplicações, ambientes de infraestrutura e outros serviços. Entre as principais características do Grafana, podemos destacar:

- Monitoramento Online.
- Visualização enquanto as aplicações são executadas com tempo em ordem de 1 segundo.
- Interatividade com interface web.
- Filtros elaborados.

Os dados coletados neste módulo são utilizados para ajudar a entender o comportamento da *fog computing* no que diz respeito à utilização de recursos de rede e computacionais, entre elas podemos citar:

- Total de carga da CPU, memória e uso de armazenamento dos *hosts* no *cluster* que hospedam o ViolET.
- Gráfico de uso da CPU dos nós da *fog*.
- Gráfico de uso de memória dos nós da *fog*.
- Gráfico de uso de entrada da rede dos nós da *fog*.
- Gráfico de uso de saída da rede dos nós da *fog*.

O monitoramento dos hosts é representado pela figura 15, apresentada a seguir.

Figura 15 – Exemplo de monitoramento dos hosts que hospedam o VIoLET

As imagens 16 e 17 ilustram exemplos das telas de monitoramento de CPU e memórias respectivamente, referente aos nós do ambiente fog.

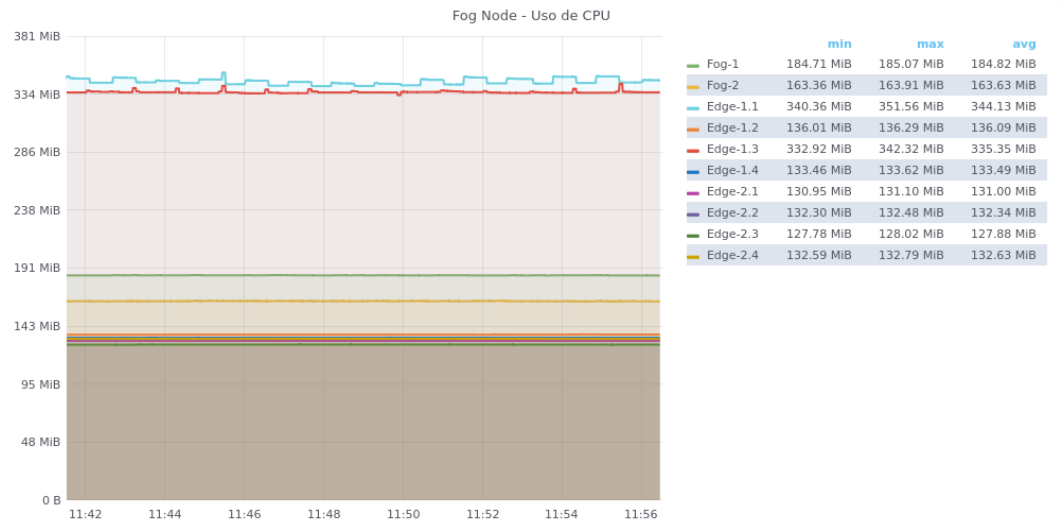


Figura 16 – Exemplo de utilização de CPU

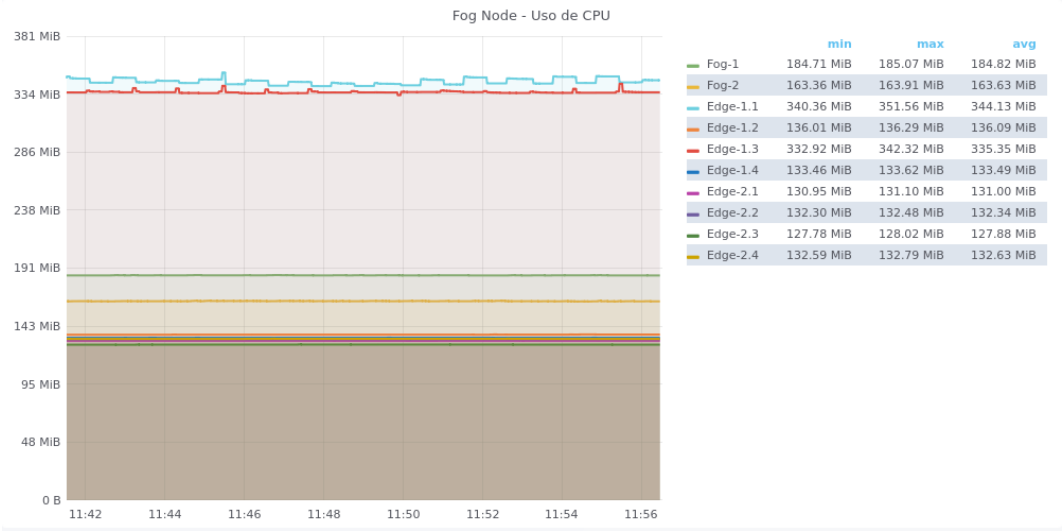


Figura 17 – Exemplo de utilização de memória

4.5.1.3 Prometheus

Prometheus (PROMETHEUS,) é uma ferramenta *open source* de monitoramento, adaptada ao atual modelo de TI e focada em serviços. Essa ferramenta foi desenvolvida pela SoundCloud em 2012 e logo seu projeto foi abraçado por outras empresas, inclusive pelo Docker, um dos principais contribuidores do projeto.

A figura 18 ilustra a integração entre as ferramentas grafana e prometheus para a coleta de métricas avaliadas no presente trabalho.

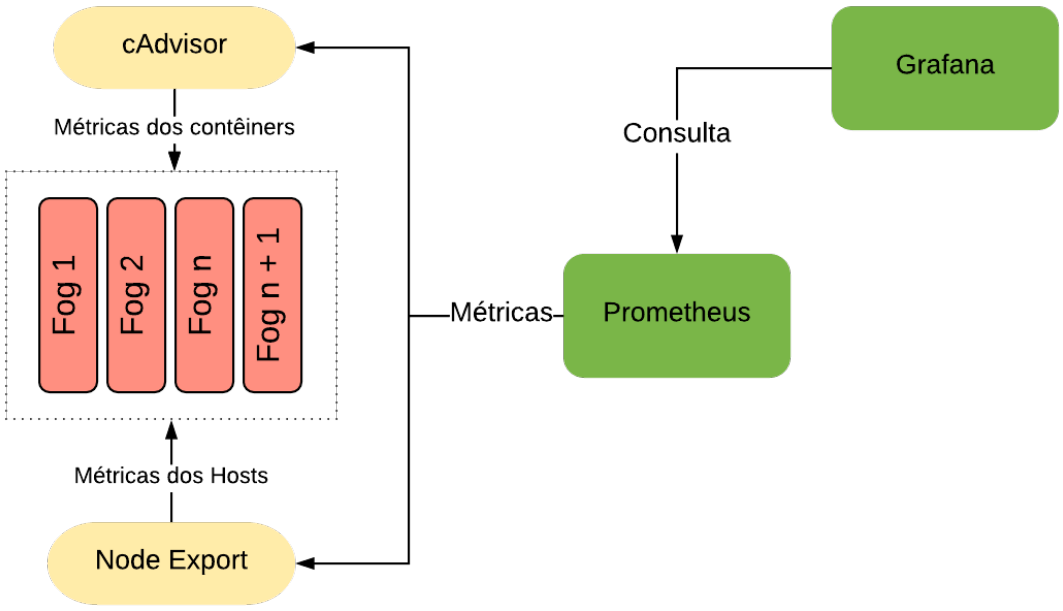


Figura 18 – Integração do monitoramento com Prometheus e Grana

Uma das características que destacam o Prometheus em relação a outros sistemas de monitoramento que estão a mais tempo no mercado, como o Zabbix e o Nagios, é que ele é um software com o foco específico em monitoramento de serviços enquanto as outras ferramentas de monitoramento estão mais relacionadas a recursos de máquina. A coleta das informações dos contêineres são feitas utilizando o cAdvisor. Por outro lado, a ferramenta Node Exporter, fica responsável pelo envio das métricas dos *hosts* para o Prometheus. For fim, o grafana coleta as informações de todos o ambiente no formato de gráficos que compõem um painel de visualização. As métricas coletadas pode servir para o envio de alertas, que podem ser definidos através do módulo Alert Manager, que faz parte do projeto Prometheus.

4.5.1.4 Portainer

O Portainer (PORTAINER,) é uma interface de usuário de gerenciamento leve e de código aberto que permite gerenciar hosts do docker ou clusters do swarm. O Portainer foi integrado ao VioLET, com o objetivo de fornecer uma interface amigável para administração do ambiente *fog* e para facilitar o acesso direto aos nós durante, mas não limitado a, configuração do ambiente para a execução dos experimentos.

4.5.1.5 Mosquitto

O projeto Mosquitto é um membro do Eclipse Foundation (LIGHT, 2017). O Mosquitto fornece implementações de servidor e cliente compatíveis com os padrões do protocolo de mensagens MQTT. O MQTT usa um modelo de publicação/assinatura, que resulta em uma baixa sobrecarga de rede e pode ser implementado em dispositivos de baixa potência, como microcontroladores que podem ser usados em sensores remotos de Internet das Coisas. Como tal, o Mosquitto é destinado ao uso em situações em que há necessidade de mensagens leves, particularmente em dispositivos restritos com recursos limitados.

A versão utilizada nesta dissertação é disponibilizada através do repositório DockerHUB e pode ser implementada facilmente em maquinas com o *docker enginer* através do seguinte comando:

docker run -d –net=rede –name mqttserver –security-opt seccomp:unconfined eclipse-mosquitto

4.5.2 Métricas

As métricas de avaliação têm como objetivo de medir o desempenho e a aplicabilidade do resultado fornecido. Além disso servem para mostrar se os resultados cumprem as metas declaradas e os critérios de sucesso. As métricas cruciais para avaliar o ambiente proposto são descritas na lista a seguir:

- **Tempo de Migração e Tempo de inatividade**

O tempo total de migração e o tempo de inatividade são muito significativos na avaliação do desempenho da migração de serviços (XAVIER et al., 2013). Estas métricas ajudam a explorar a eficiência da estratégia indicando o tempo total de migração e tempo de inatividade enquanto a migração ocorre.

- **Tempo de resposta entre cliente e aplicação como medida de tempo em (ms) e largura de banda fornecida em (Mbps)**

  Usamos RTT (*Round Trip Time*) como a métrica de latência e medimos a largura de banda de *uplink* e *downlink*.

- **Uso de CPU e memória como medida de eficiência para uso de recursos computacionais.**

4.5.3 Carga de Trabalho

Um dos principais impulsionadores da *fog computing* para a IoT tem sido a necessidade de baixa latência e maior escalabilidade concomitante com a proliferação dos dispositivos e sensores. As aplicações IoT tendem a ter os sensores como fontes de dados, e tais dados geralmente estão em forma de tuplas (BADIGER; BAHETI; SIMMHAN, 2018). Além disso, o fluxo de dados gerado envolve o uso de recursos de IoT para capturar dados, recursos do ambiente *fog* para executar o processamento inicial dos dados e um *data center* em nuvem para trabalhos intensivos em computação.

Os protocolos para a comunicação entre os componentes da IoT devem lidar com fatores como baixa largura de banda, alta latência e instabilidade da comunicação. Alguns protocolos foram criados exatamente para lidar com tais fatores: CoAP (*Constrained Application Protocol* (SHELBY; HARTKE; BORMANN, 2014)), MQTT (*Message Queuing Telemetry Transport*(BANKS; GUPTA, 2014)), dentre outros. Neste trabalho, o protocolo MQTT foi adotado para a realizacão dos experimentos, devido a ampla gama de *brokers* implementados em diferentes linguagens e a crescente adoção no mercado. O protocolo MQTT define dois tipos de entidades na rede: um *message broker* e inúmeros clientes. O *broker* é um servidor que recebe todas as mensagens dos clientes e, em seguida, roteia essas mensagens para os clientes de destino relevantes. Um cliente é qualquer coisa que possa interagir com o *broker* e receber mensagens. Um cliente pode ser um sensor de IoT em campo ou uma aplicação em um *data center* que processa dados de IoT.

Em um primeiro momento, a carga é dada através do envio de dados de clientes para *brokers* MQTT alocados em diferentes servidores do ambiente *fog* de maneira distribuída, considerando os recursos de computação disponíveis desses servidores, como CPU, memória, armazenamento e informações da rede.

5
Resultados e Discussão

Nesse capítulo, discutimos a importância da proposta, analisando os resultados e comparando-os com diferentes estratégias de alocação de recursos.

5.1 Análise da utilização de recursos da solução de Cluster e Orquestração

Uma das principais limitações em um ambiente *fog computing* é o poder computacional dos seus dispositivos. Assim, a adoção de soluções para proporcionar a virtualização e orquestração dos recursos na *fog computing* requer componentes que funcionem utilizando o mínimo possível de recursos sem comprometer o desempenho do ambiente e a qualidade de serviço para as aplicações IoT. Com esse objetivo, conduzimos um experimento para identificar a quantidade de recursos necessário a cada dispositivo *fog computing* para atender à solução apresentada.

A análise dos dados é representada neste trabalho através de gráficos de caixa ou *box plot*. O gráfico de caixa é um método alternativo ao histograma e ao ramo-e-folha para representar os dados. Em um gráfico de caixa são apresentados 5 estatísticas: o mínimo, o primeiro quartil (Q1), a mediana, o terceiro quartil (Q3) e o máximo. Além disso, apresenta também os *outrliers*, ou seja, pontos desgarrados podem afetar de forma adversa as decisões a serem tomadas a partir da análise dos dados se não forem devidamente considerados.

O gráfico de caixa ilustrado no figura 19 representa a utilização de CPU durante o monitoramento e execução do serviço de orquestração conectando os *gateways* (Fog-1 e FOg-2) aos seus respectivos nós de borda (Edges-1.1, 1.2, 1.3 e 1.4, 2.1, 2.2, 2.3, 2.4) na *fog computing*. Para efeitos de comparação os nós de borda conectados ao Fog-2 não fazem parte do *cluster*, o objetivo deste experimento é comparar o impacto da solução proposta em relação à utilização de recursos.

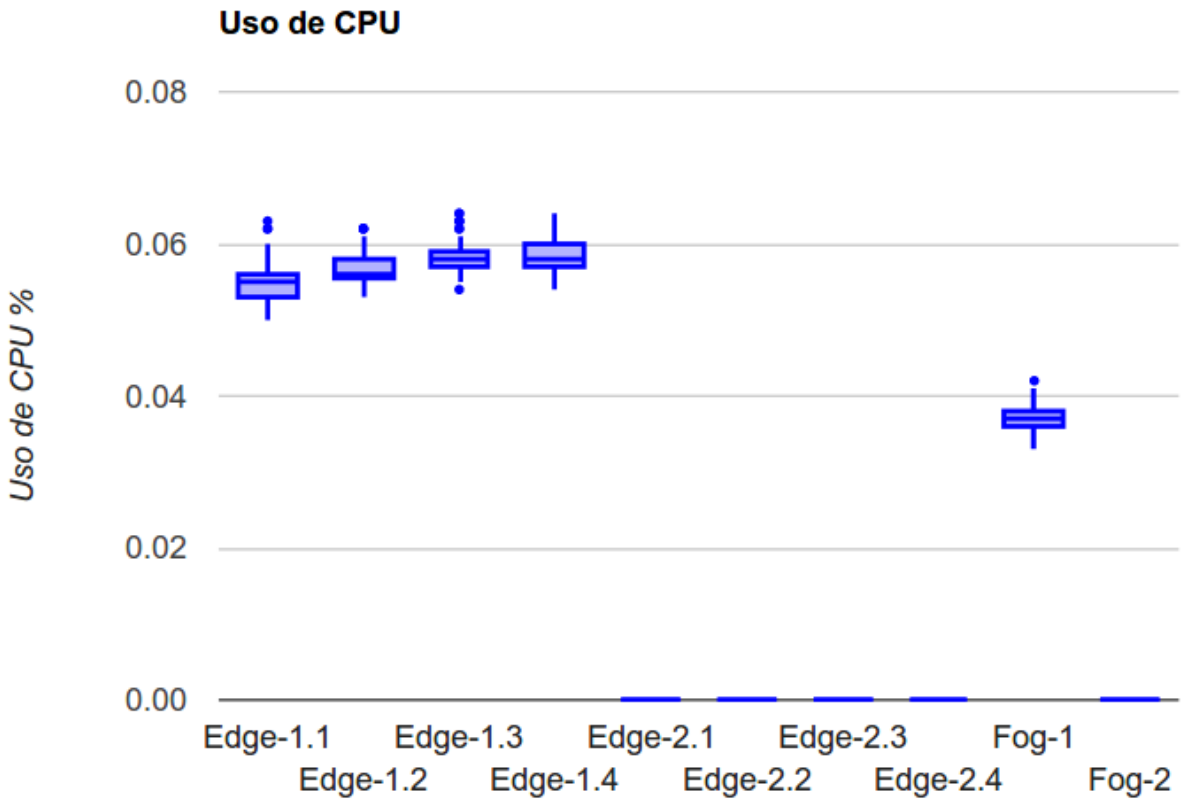


Figura 19 – Utilização de CPU

A taxa de utilização de CPU dos nós da *fog computing* conectados ao Docker Swarm apresentam uma taxa média de utilização próxima de 0.06%, esse valor corresponde ao consumo dos nós Edge (1.1, 1.2, 1.3 e 1.4). O nó Fog-1 apresenta um consumo de CPU menor que os demais da sua rede privada, com o valor de 0.04%, isto ocorre devido ao maior poder de processamento do dispositivo. Os demais nós (Fog-2 e os Edges (2.1, 2.2, 2.3 e 2.4)) apresentam uma taxa de utilização mínima, próxima do 0%, mas isto acontece porque o dispositivos não fazem parte do *cluster* neste momento do experimento.

O gráfico de caixa ilustrado na figura 20 representa a taxa de utilização de memória para cada dispositivo na *fog computing*. O nó Fog-1 apresenta uma maior taxa de utilização de memória, cerca de 93.75MB, enquanto os quatro nós Edge (1.1, 1.2, 1.3 e 1.4), permanecem com uma taxa utilização média de 75MB. A diferença de utilização entre os nós Edge (1.1, 1.2, 1.3 e 1.4) e o Fog-1, dá-se devido ao componente de orquestração do *cluster*, chamado *Manager* estar operando no nó Fog-1, o que faz com que o mesmo consuma mais memória que os demais. Por outro lado o Fog-2 resultou em um consumo médio de memória de 23MB e seus respectivos nós Edge, uma média de 20MB, estes valores correspondem ao consumo para atender as necessidades básicas do sistema operacional.

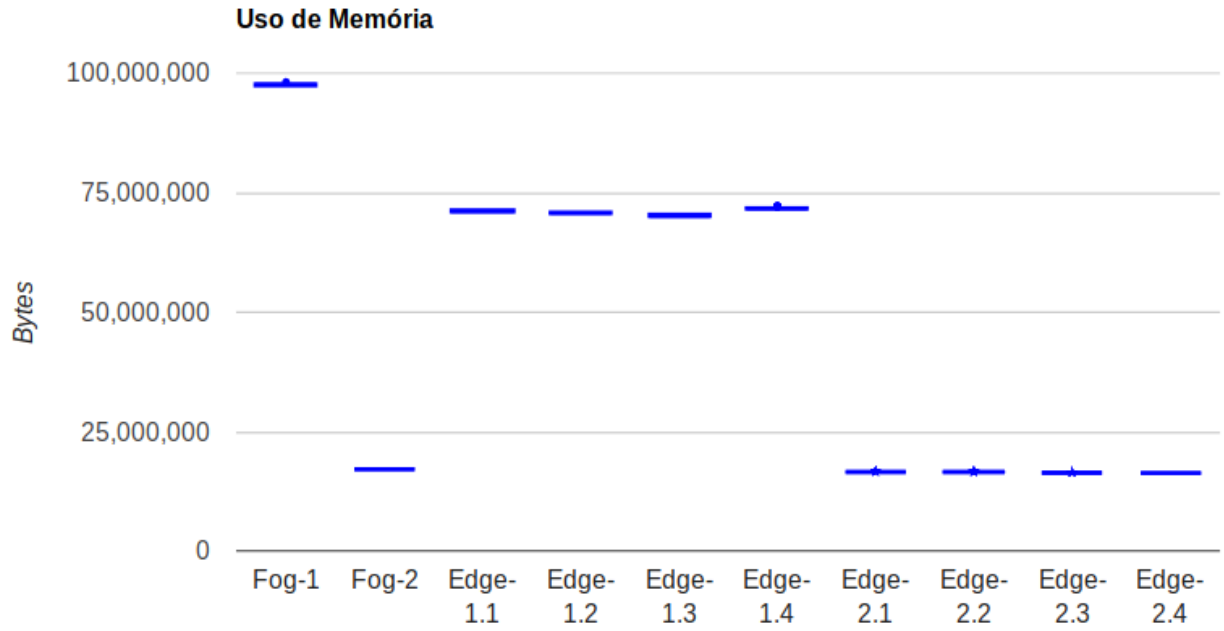


Figura 20 – Utilização de Memória

Em relação à transferência de dados na rede, as figuras 21 e 22 ilustram o tráfego de rede de entrada e saída respectivamente. O gráfico do tráfego de entrada indica uma utilização média de 632B no dispositivo Fog-1, enquanto os demais nós da rede privada 1 apresentam um utilização média de 430B. Este tráfego inclui valores para orquestração e monitoramento.

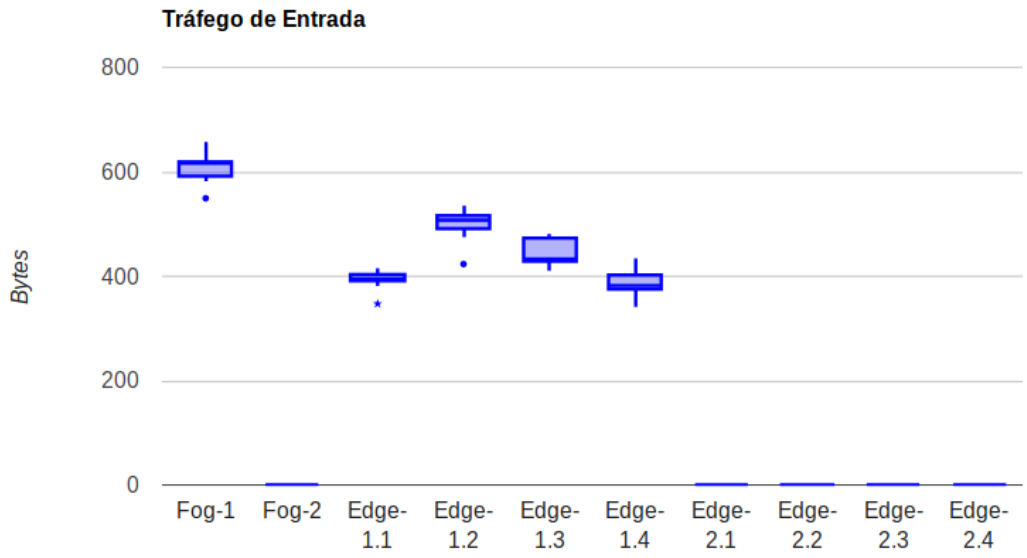


Figura 21 – Tráfego de Entrada

De forma análoga, a figura 22 ilustra o tráfego de saída durante a execução do Docker Swarm. O nó Fog-1 apresenta uma taxa média de utilização de 600B e os demais nós utilizam

em média 440B.

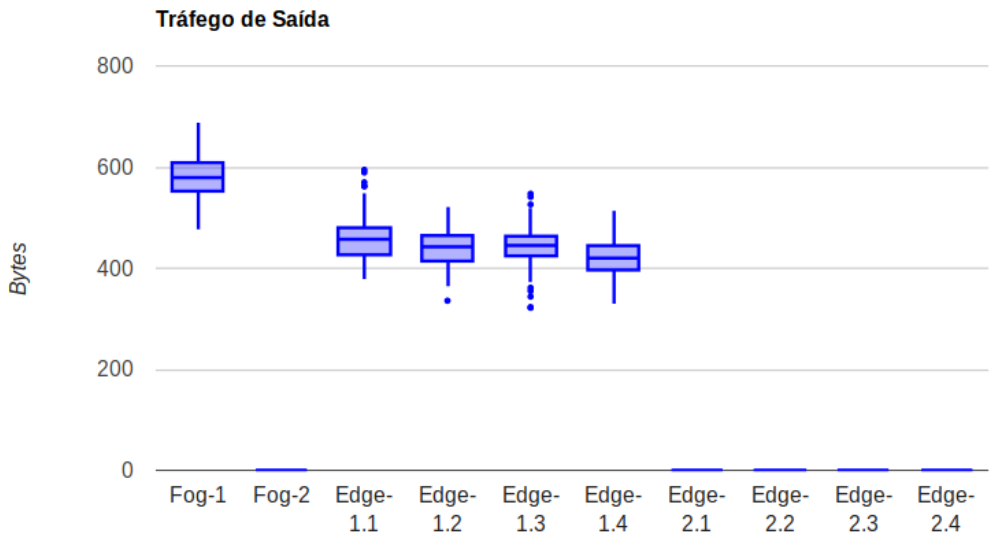


Figura 22 – Tráfego de Saída

Adicionalmente, conduzimos um experimento para validar as características de rede definidas no ambiente do experimento. Os resultados são apresentados na tabela 12.

Tabela 12 – Validação das Características de rede

Rede	**Resultados Esperados**		**Resultados**	
	Latência esperada (ms)	**Largura de banda (Mbps)**	**Latência (ms)**	**Largura de Banda (Mbps)**
Rede Privada 1	2	100	2	97,3
Rede Privada 2	4	75	4	73,66
Rede Pública	100	40	100	38,3

Observamos através dos resultados apresentados na tabela 12, que o cenário de rede desenvolvido no VIolET apresenta valores de latência e largura de banda próximos aos especificados na configuração do ambiente, o que representa uma maior acurácia e confiabilidade nos resultados dos experimentos realizados.

A seguir apresentamos os resultados iniciais relacionados à técnica de migração de serviço em uma *fog computing*.

5.2 Migração de Serviços

O principal objetivo do experimento é investigar o desempenho da migração ao vivo e suas limitações em um ambiente *fog computing*. O tempo total de migração e o tempo de inatividade durante o processo de migração de contêineres entre nós distintos são utilizados como métrica para reproduzir uma versão simplificada da estratégia apresentada na seção 4.

Primeiro, analisamos o tempo de resposta durante a execução de uma aplicação cliente/servidor MQTT em relação à execução do algoritmo com e sem a otimização. Os dados do sensores são gerados através de um *script* escrito em linguagem *Python* que simula sensores de temperatura e umidade com o objetivo de simular um cenário de *Smart Cities*. Para simplificar o experimento, os dados de sensores são enviados através do nó de borda (Edge-1.1). Neste experimento o servidor MQTT é alocado e migrado entre diferentes nós utilizando a estratégia descrita no algoritmo de First-Fit. Em seguida, é aplicada um restrição no algoritmo com o objetivo de limitar os parâmetros de QoS, onde a latência do nó escolhido deve corresponder a um valor menor que 10ms para que o nó possa ser escolhido de acordo com o cenário proposto, conforme valores apresentados na tabela 9. Os testes foram executados durante um período aproximado de 35 minutos com vinte repetições cada e os resultados do são ilustrados na figura 23.

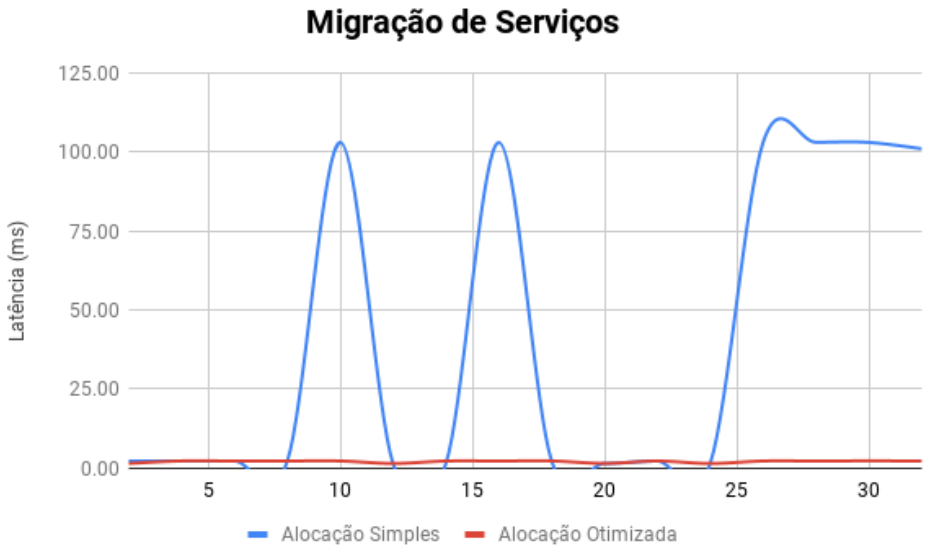


Figura 23 – Latência entre cliente e servidor MQTT

Conforme apresentado na figura 23, o cenário de execução com o algoritmo sem a estratégia de otimização (Alocação Simples) obteve valores consistentes, porém desalinhados em relação à continuidade de uma baixa latência durante o provimento do serviço. Neste caso, o algoritmo não verificou questões de restrição de rede quanto à alocação dos nós selecionados.

Por outro lado, a estratégia de otimização aplicada no algoritmo (Estratégia Otimizada), priorizou a utilização de nós disponíveis mas que atendiam as restrições de latência impostas. É digno de nota que, a depender das necessidades da aplicação, políticas de alocação de recursos podem ser definidas para atender diferentes níveis de QoS, como por exemplo, priorizar a alocação de módulos na borda da rede que necessitam de menor latência ou migrar módulos de nós que eventualmente ultrapassem limites mínimos ou máximos da utilização de recursos estipulados no ambiente.

De acordo com os resultados ilustrados na figura 24 e detalhados na tabela 13, observamos

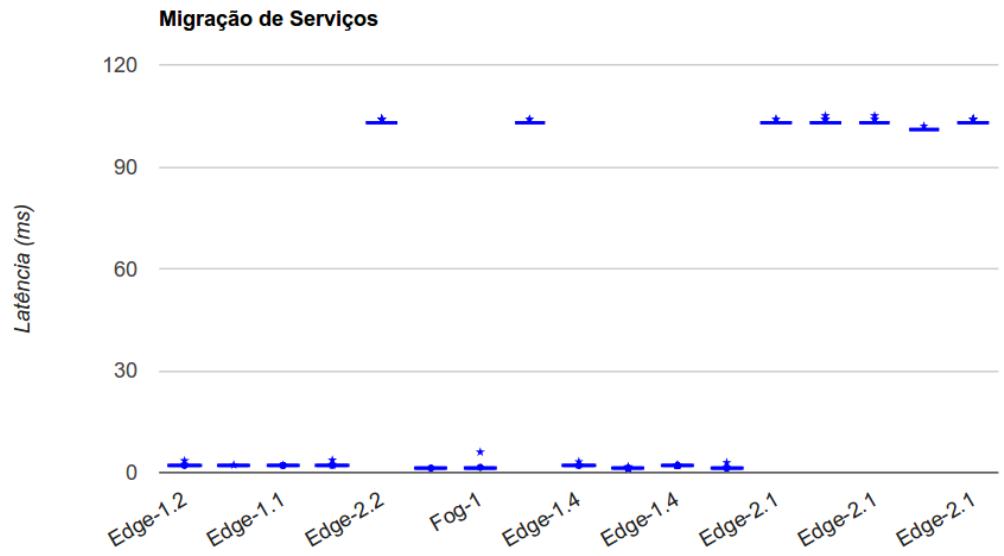


Figura 24 – Latência durante migração dos serviço MQTT

que à medida em que a aplicação é migrada para nós da rede privada 2, o tempo de resposta aumentou consideravelmente. Isto é esperado, uma vez que os nós encontram-se mais distantes dos sensores. Além disso, a gráfico de caixa apresenta números pouco expressivos de *outliers*, na mostra, o que indica uma maior consistência nos valores apresentados. É importante observar que em cenários reais, as aplicações podem enfrentar problemas ao tentar garantir qualidade de serviço fim-a-fim, uma vez que os pacotes podem ser enviados através da internet, o que dificulta a gestão da rede como um todo.

Tabela 13 – Alocação Simples

Sequência	fog computing node	Latência (ms)	Desvio Padrão	Variância	MIN	MAX
1	Edge-1.2	2,15	0,1302988802	0,01726733304	2,08	3,51
2	Edge-1.4	2,14	0,02122277184	0,0004580979284	2,10	2,28
3	Edge-1.1	2,15	0,021765517	0,0004712516297	2,09	2,22
4	Edge-1.3	2,17	0,1648843571	0,02763696943	2,07	3,72
5	Edge-2.2	103,08	0,2944775709	0,08815007968	103,00	105,00
6	Fog-1	1,35	0,05654044798	0,003245581631	1,21	1,64
7	Fog-1	1,40	0,4361407846	0,1934573953	1,22	6,08
8	Edge-2.4	103,03	0,1567626732	0,02498913516	103,00	104,00
9	Edge-1.4	2,17	0,200230818	0,04077117195	2,09	4,02
10	Fog-1	1,37	0,06284545795	0,004017043315	1,20	1,73
11	Edge-1.4	2,14	0,01854802891	0,0003481095176	2,07	2,20
12	Fog-1	1,36	0,1705833091	0,02959177169	1,22	2,95
13	Edge-2.1	103,02	0,1285467104	0,01680428799	103,00	104,00
14	Edge-2.3	103,08	0,2944775709	0,08815007968	103,00	105,00
15	Edge-2.1	103,03	0,2220141985	0,05012313487	103,00	105,00
16	Fog-2	101,02	0,1285467104	0,01680428799	101,00	102,00
17	Edge-2.1	103,04	0,2006275023	0,04092423584	103,00	104,00

Por outro lado, os resultados ilustrados na figura 25 e detalhados na tabela 14, representam

os valores de latência obtidos no experimento quando aplicada a técnica de otimização.

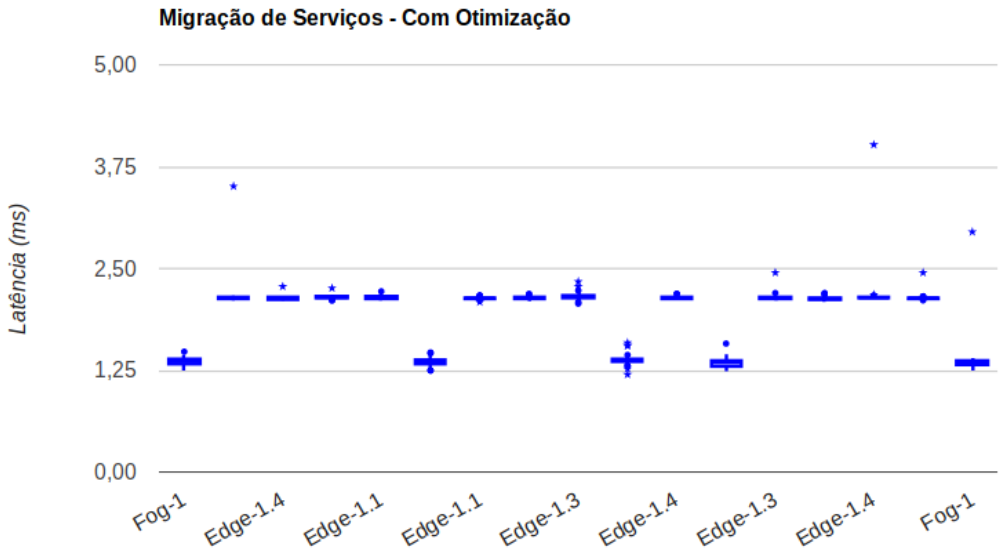


Figura 25 – Latência durante migração dos serviço MQTT - Estratégia Otimizada

Tabela 14 – Alocação com Otimização

Sequência	fog computing Node	Latência (ms)	Desvio Padrão	Variância	MIN	MAX
1	Fog-1	1,36	0,04573226333	0,002142450639	1,25	1,48
2	Edge-1.2	2,17	0,2091929205	0,044829036	2,11	3,51
3	Edge-1.4	2,14	0,02670914983	0,0007307781649	2,11	2,28
4	Edge-1.3	2,15	0,02372843334	0,0005767711963	2,11	2,26
5	Edge-1.1	2,15	0,02212620982	0,0005015098722	2,11	2,22
6	Fog-1	1,35	0,05188291371	0,002757491289	1,25	1,47
7	Edge-1.1	2,14	0,01531972185	0,0002404181185	2,09	2,17
8	Edge-1.2	2,14	0,01792684808	0,0003292102207	2,10	2,19
9	Edge-1.3	2,16	0,04423960733	0,002004878049	2,07	2,34
10	Fog-1	1,38	0,06506232839	0,004336353078	1,20	1,59
11	Edge-1.4	2,14	0,01572870911	0,0002534262485	2,12	2,19
12	Fog-1	1,34	0,05904569889	0,003571428571	1,24	1,58
13	Edge-1.3	2,15	0,05010193691	0,002571428571	2,11	2,45
14	Edge-1.4	2,14	0,02204535626	0,0004978513357	2,09	2,20
15	Edge-1.4	2,19	0,2862084021	0,08391318235	2,13	4,02
16	Edge-1.2	2,14	0,04909382936	0,002468989547	2,11	2,45
17	Fog-1	1,38	0,2487611026	0,06339140534	1,25	2,95

O gráfico de caixa representado pela figura 26 ilustra o tempo de execução de cada passo percorrido durante o processo de migração dos serviços entre dois nós. Por questões de simplificação, utilizamos como referência o tempo de migração entre os nós Fog-1 e Edge-1.2.

Conforme resultados apresentados na tabela 15, a migração total levou um tempo médio de 9.88 segundos para a execução completa. A fase "Baixar imagem", foi a que levou mais tempo durante a migração, isto compreende processo de baixar a imagem completa de um

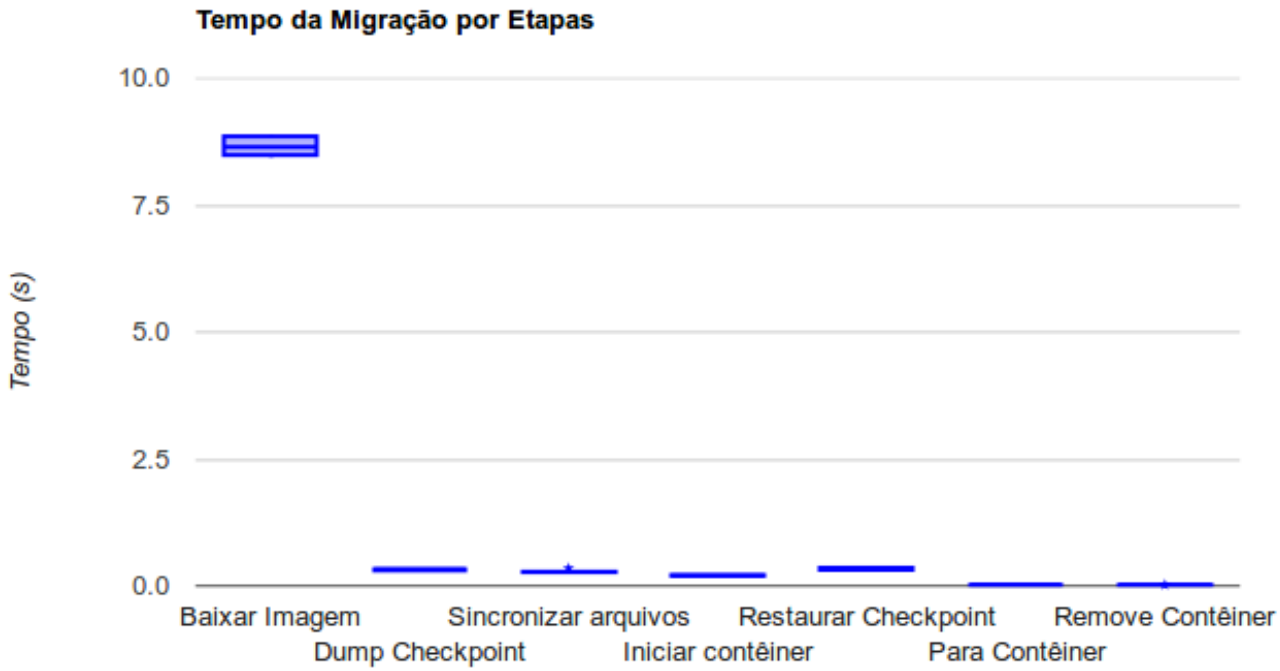


Figura 26 – Tempo de Migração

repositório para o nó de destino para que em seguida, o *checkpoint* seja restaurado. As demais fases compreendem o passo para envio dos dados, restauração do *checkpoint* e finalização do processo.

Tabela 15 – Tempo de Execução de Migração

	Fase	Tempo médio (s)	Desvio Padrão	Variância	MIN	MAX
1	Baixar Imagem	8,6602	0,1995	0,0398	8,4450	8,8718
2	Dump Checkpoint	0,3247	0,0148	0,0002	0,3084	0,3398
3	Sincronizar arquivos	0,2938	0,0332	0,0011	0,2753	0,3527
4	Iniciar Contêiner	0,2140	0,0079	0,0001	0,2067	0,2252
5	Restaurar Checkpoint	0,3407	0,0294	0,0009	0,3100	0,3794
6	Parar Contêiner	0,0251	0,0030	0,0000	0,0229	0,0299
7	Remover Contêiner	0,0242	0,0011	0,0000	0,0234	0,0262
	Total	**9,8828**	**0,2889**	**0,0421**	**9,5917**	**10,2250**

Durante a migração, o tempo de indisponibilidade da aplicação compreende o tempo entre o inicio da fase 3 e o término da fase 5. Logo após finalizar o *dump* do *checkpoint*, o serviço no nó de origem é finalizado. Quando o nó de destino inicia o contêiner com o mesmo nome contido no nó de origem, a rede *overlay* do Docker atribui o mesmo endereço IP. Isto permite que quando ocorra a migração do serviços entre os nós, o cliente da aplicação continue associado

ao mesmo endereço, garantindo alta disponibilidade e resiliência no ambiente.

A seguir, a tabela 16, apresenta um resumo das características e resultados da migração.

Tabela 16 – Características da Migração

Imagem		esclipse-mosquitto
Tamanho da Imagem (MB)		4,37
Tamanho do Chekpoint (KB)		178,94
Indisponibilidade (s)	Tempo Total	2,32
	Desvio Padrão	0,062
	Variância	1,42
	Valor Máximo	2,38
	Valor Mínimo	2,18
Migração (s)	Tempo Total	9,88275
	Desvio Padrão	0,18
	Variância	0,0405
	Valor Máximo	10,0930
	Valor Mínimo	9,6102

Em linhas gerais, os resultados iniciais obtidos neste trabalho confirmam a hipótese de que a otimização do ambiente pode ser alcançada através da adoção de estratégias eficientes para a escolha dos nós e migração dos serviços, que utilizem como base informações relevantes do ambiente orquestrado para a tomada de decisões como resposta para a questão de pesquisa: "Como os serviços podem ser alocados através dos nós de um ambiente *fog computing computing* de modo a proporcionar a otimização da utilização de recursos?"

No capítulo a seguir, apresentamos a conclusão do trabalho e linhas gerais dos desafios futuros e direções de pesquisas na área.

6
Conclusão

Neste capítulo são apresentadas as principais conclusões relacionadas ao trabalho desenvolvido, assim como as publicações realizadas e os trabalhos futuros.

6.1 Principais Conclusões

Esta dissertação de mestrado apresenta a *fog computing* como uma alternativa para o provimento de qualidade de serviços para aplicações sensíveis à latência. No presente trabalho apresentamos o conceito da *fog computing*, seus desafios, sua contextualização teórica e os trabalhos correlatos através de um mapeamento sistemático. Em contraste com as soluções tradicionais, descrevemos um modelo de arquitetura autonômica como alternativa para o problema de alocação de recursos de infraestruturas. Com base no modelo proposto, apresentamos um mecanismo de migração de serviços entre os nós da *fog* e utilizando contêineres Docker e CRIU (*Checkpoint Live Restore*) para dar suporte a implementação de estratégias de otimização. Nessa estratégia diferentes objetivos podem ser alcançados, tais como, prover economia de energia, diminuir a latência, ou otimizar a utilização de recursos de acordo com as necessidades de QoS de cada tipo de aplicação.

Os testes foram realizados em um ambiente virtual de larga escala para Internet das Coisas, chamado VioLET. Os resultados preliminares apresentaram um tempo médio de migração de 9.88 segundos com um *downtime* próximo aos 2 segundos. Estes resultados são promissores, uma vez que torna possível manter alta disponibilidade e curtos períodos de indisponibilidade durante a migração dos serviços. Através dos testes, demonstrou-se que a *Fog Computing* pode se beneficiar com a migração de contêineres para a execução de diversas aplicações IoT. Isto confirma a hipótese de que é possível prover QoS para aplicações IoT adequando os objetivos da técnica de otimização considerando as vantagens em termos de baixa latência e alta largura de banda.

Com o rápido crescimento da IoT e das aplicações sensíveis à latência, acreditamos que este trabalho de pesquisa contribuirá para o desenvolvimento de novas técnicas para o gerenciamento eficiente de recursos, especialmente em termos de latência, desempenho e eficiência energética. Por fim, concluímos o trabalho apresentando direções de pesquisa em linhas gerais para trabalhos futuros. As ferramentas e *scripts* utilizados neste trabalho, bem como os códigos da arquitetura estão disponíveis em https://github.com/danilosilvase/autonomic.

6.2 Publicações Realizadas

Nesta seção encontram-se descritos os artigos publicados no andamento desta dissertação de mestrado. Os artigos contemplam os principais esforços alcançados até o momento, bem como sua projeção nas pesquisas realizadas junto ao Grupo de Pesquisa em Redes e Computação Distribuída (GPRCom) na Universidade Federal de Sergipe.

- MACHADO, J.; **SOUZA, D.**; SILVA, R.; MENEZES, A.; MORENO, E.; RIBEIRO, A. Avaliação de Performance para Fornecer StaaS a Dispositivos IoT em Ambiente Fog Computing. XIX Simpósio de Sistemas Computacionais de Alto Desempenho. São Paulo-SP, Brasil: WSCAD, 2018.
- SANTO, W; **SOUZA, D.**; MORENO, E.; RIBEIRO, A. Internet of Things: A Survey on Communication Protocol Security. 9th Euro American Conference on Telematics and Information Systems. Fortaleza-CE, Brasil: EATIS 2018.

6.3 Publicações Submetidas

- CARVALHO, R; **SOUZA, D.**; SANTO, W.; RIBEIRO, A.; MORENO, E. Machine Learning Algorithms to Detect Ddos Attacks in SDN. International Journal of Communication Systems (2018).
- **SOUZA, D.**; MACHADO, J.; A.; MORENO, E.; RIBEIRO, A. Service Migration and High Availability of Distributed Application on Fog Computing Environments. International Journal of Web and Grid Services (IJWGS), 2018.
- **SOUZA, D.**; MACHADO, J.; A.; MORENO, E.; RIBEIRO, A. Mapeamento Sistemático sobre Gerenciamento de Recursos em Ambiente Fog Computing. Revista de Informática Teórica e Aplicada. Porto Alegre-RS, Brasil: RITA 2018.
- MACHADO, J.; **SOUZA, D.**; RIBEIRO, A.; MORENO, E. FOGSYS – A System for the Implementation of StaaS Service in a Fog Computing using Embedded Platforms. Concurrency and Computation: Practice and Experience, 2018.

- MACHADO, J.; **SOUZA, D.**; RIBEIRO, A.; MORENO, E. Performance Analysis to Provide StaaS to IoT Devices in Fog Computing Environment Using Embedded Systems . Concurrency and Computation: Practice and Experience, 2018.
- **SOUZA, D.**; MACHADO, J.; A.; MORENO, E.; RIBEIRO, A. Uma Estratégia Autonômica para Alocação de Recursos em Ambientes Fog Computing. Revista de Informática Teórica e Aplicada. Porto Alegre-RS, Brasil: RITA 2018.
- MACHADO, J.; **SOUZA, D.**; RIBEIRO, A.; MORENO, E. Análise de Performance para Fornecer StaaS a Dispositivos IoT em Ambiente Fog Computing Utilizando Sistemas Embarcados. IEEE Latino Americano, 2018.
- MACHADO, J.; **SOUZA, D.**; RIBEIRO, A.; MORENO, E. Um Estudo dos Algoritmos de Criptografia Leve Baseado em Internet das Coisas. Revista de Informática Teórica e Aplicada. Porto Alegre-RS, Brasil: RITA 2018.
- MACHADO, J.; **SOUZA, D.**; RIBEIRO, A.; MORENO, E. FogSys: Sistema para Implementação da Fog Computing para Fornecer StaaS a Dispositivos IoT. Revista de Informática Teórica e Aplicada. Porto Alegre-RS, Brasil: RITA 2018.

6.4 Trabalhos Futuros

Este trabalho foi desenvolvido com o objetivo de permitir a otimização da utilização de recursos de um ambiente *fog computing* e prover qualidade de serviço para o uso de aplicações sensíveis à latência. Contudo, apesar dos esforços recentes em pesquisa delineados neste trabalho, muitos desafios ainda permanecem em aberto. Dessa forma, amplia-se a possibilidade de estender este trabalho, sendo:

- **Implementar técnicas de autenticação e segurança**. Manter segurança em ambientes de computação em *fog* é um desafio consideravelmente difícil, uma vez que seus componentes localizam-se principalmente na borda da rede próximo ao dispositivos e sensores. Não obstante, embora seja possível utilizar técnicas semelhantes às aplicadas na nuvem em um ambiente *fog*, as pesquisas na área ainda se encontram em seu estágio inicial. No entanto, na literatura existente, as preocupações de segurança na computação *Fog* foram discutidas em termos de autenticação de usuários, privacidade, troca segura de dados, ataque DoS, etc. Outros aspectos são igualmente importantes, tais como, segurança e privacidade são abordados no trabalhos de (YI; QIN; LI, 2015) e (STOJMENOVIC; WEN, 2014).
- **Propor novas técnicas para diminuir o tempo de migração dos serviços.** Um dos desafios que a mobilidade impõe às plataformas de IoT é a capacidade de fornecer e manter os recursos de computação e armazenamento próximos das "coisas"e, no caso de uma mudança repentina no local, o ambiente deve ser capaz de reconfigurar o contexto

de comutação e orquestrar a alocação de recursos e a migração do estado para um novo local. As técnicas de virtualização de recursos baseadas em contêineres são candidatos em potencial para permitir migrações de serviços de forma rápida, movendo apenas o estado do contêiner. Tecnologias como, *Docker* (MERKEL, 2014) e *CRIU* (YANG, 2015), são exemplos representativos, mas definir técnicas para a migração de serviços entre dispositivos da *fog* e a nuvem ainda é um desafio a ser superado. Além disso, faz-se necessário propor técnicas para diminuir o tempo total de migração de serviços.

- **Avaliar diferentes técnicas de virtualização**. Os modelo de virtualização de serviços disponibilizados através de nós *fog computing* também é um desafio a ser superado. Rede Definidas por Software (SDN) é considerada como um dos principais facilitadores da rede virtualizada. SDN resume-se em uma arquitetura que permite que o controle de rede se torne diretamente programável e a infraestrutura subjacente seja abstraída para as aplicações e implementadas em servidores separados. Um dos aspectos importantes do SDN é fornecer suporte para NFV. NFV é um conceito de arquitetura que virtualiza as funções tradicionais de rede tais como, conversão de endereços de rede (NAT), *firewall*, detecção de intrusões, serviço de nomes de domínio (DNS), etc.

- **Otimizar o consumo de energia utilizando técnicas DVFS em um ambiente real.** Os nós da *fog* precisam lidar com um grande número de solicitações simultâneas de serviço provenientes dos dispositivos IoT. Uma das soluções triviais é implantar os nós *fog* no ambiente de acordo com a demanda. No entanto, esta abordagem aumentará em grande medida o número de nós *fog* ativos, afetando o consumo total de energia do sistema. Técnicas de DVFS são bem consolidados em ambientes de computação em nuvem, contudo, o uso desta técnica também podem ser um campo potencial de pesquisa baseada em *fog computing*.

- **Comparar diferentes estratégias de alocação recursos utilizando algoritmos evolucionários.** A maioria dos estudos sobre otimização de recursos abordam um único critério. No entanto, muitos problemas do mundo real, exigem a consideração de vários critérios. Por esta razão, pesquisas recentes tendem a olhar para situações de múltiplos objetivo.

- **Avaliar o desempenho do ambiente *fog* utilizando a ferramentas de *benchmark***. Definir métodos para avaliar ambientes IoT através de ferramentas de *benchmark* é uma tarefa importante, principalmente no que se diz respeito à avaliação de técnicas de gerenciamento. Ao melhor do nosso conhecimento, são poucos os trabalhos nesta área. Como exemplo podemos citar, RIoTBench (SHUKLA; CHATURVEDI; SIMMHAN, 2017), um conjunto de micro *benchmarks* desenvolvidos para avaliar sistemas de processamento de fluxo distribuído para aplicações IoT.

Referências

AAZAM, M.; HUH, E.-N. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In: IEEE. *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*. [S.l.], 2015. p. 687–694. Citado 6 vezes nas páginas 21, 36, 40, 42, 44 e 45.

AAZAM, M. et al. Mefore: Qoe based resource estimation at fog to enhance qos in iot. In: IEEE. *Telecommunications (ICT), 2016 23rd International Conference on*. [S.l.], 2016. p. 1–5. Citado 2 vezes nas páginas 39 e 42.

AGARWAL, S.; YADAV, S.; YADAV, A. K. An efficient architecture and algorithm for resource provisioning in fog computing. *International Journal of Information Engineering and Electronic Business*, Modern Education and Computer Science Press, v. 8, n. 1, p. 48, 2016. Citado 2 vezes nas páginas 40 e 44.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado 4 vezes nas páginas 8, 14, 20 e 21.

BADIGER, S.; BAHETI, S.; SIMMHAN, Y. Violet: A large-scale virtual environment for internet of things. *arXiv preprint arXiv:1806.06032*, 2018. Citado 4 vezes nas páginas 17, 57, 58 e 63.

BAJPAI, A.; CHOUDHURY, B.; CHOUDHURY, S. An adaptive and elastic cloud based framework for service oriented computing in internet of things. In: IEEE. *Communication Systems and Networks (COMSNETS), 2017 9th International Conference on*. [S.l.], 2017. p. 460–463. Citado 6 vezes nas páginas 37, 39, 40, 42, 44 e 45.

BANKS, A.; GUPTA, R. Mqtt version 3.1. 1. *OASIS standard*, v. 29, 2014. Citado na página 63.

BELOGLAZOV, A. et al. Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services. *Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, the University of Melbourne, Australia*, 2016. Citado na página 39.

BONOMI, F. et al. Fog computing: A platform for internet of things and analytics. In: *Big Data and Internet of Things: A Roadmap for Smart Environments*. [S.l.]: Springer, 2014. p. 169–186. Citado 3 vezes nas páginas 14, 26 e 47.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: ACM. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. [S.l.], 2012. p. 13–16. Citado na página 24.

BRUN, Y. et al. Engineering self-adaptive systems through feedback loops. *Software engineering for self-adaptive systems*, Springer, v. 5525, p. 48–70, 2009. Citado na página 29.

CARDELLINI, V. et al. On qos-aware scheduling of data stream applications over fog computing infrastructures. In: IEEE. *Computers and Communication (ISCC), 2015 IEEE Symposium on*. [S.l.], 2015. p. 271–276. Citado 6 vezes nas páginas 35, 38, 39, 42, 44 e 45.

CARDELLINI, V. et al. Optimal operator replication and placement for distributed stream processing systems. *ACM SIGMETRICS Performance Evaluation Review*, ACM, v. 44, n. 4, p. 11–22, 2017. Citado na página 42.

CHARALAMPIDIS, P.; TRAGOS, E.; FRAGKIADAKIS, A. A fog-enabled iot platform for efficient management and data collection. In: IEEE. *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2017 IEEE 22nd International Workshop on*. [S.l.], 2017. p. 1–6. Citado 3 vezes nas páginas 40, 42 e 44.

COMPUTING, A. et al. An architectural blueprint for autonomic computing. *IBM White Paper*, v. 31, 2006. Citado 5 vezes nas páginas 8, 15, 28, 29 e 50.

COMPUTING, F. *the Internet of Things: Extend the Cloud to Where the Things Are*. [S.l.]: Cisco Syst., San Jose, CA, USA, 2016. Citado na página 15.

DANTZIG, G. *Linear programming and extensions*. [S.l.]: Princeton university press, 2016. Citado na página 27.

DASTJERDI, A. V.; BUYYA, R. Fog computing: Helping the internet of things realize its potential. *Computer*, IEEE, v. 49, n. 8, p. 112–116, 2016. Citado 2 vezes nas páginas 14 e 17.

DENG, R. et al. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, IEEE, v. 3, n. 6, p. 1171–1181, 2016. Citado 4 vezes nas páginas 39, 40, 42 e 44.

DO, C. T. et al. A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing. In: IEEE. *Information Networking (ICOIN), 2015 International Conference on*. [S.l.], 2015. p. 324–329. Citado 2 vezes nas páginas 39 e 44.

DSOUZA, C.; AHN, G.-J.; TAGUINOD, M. Policy-driven security management for fog computing: Preliminary framework and a case study. In: IEEE. *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*. [S.l.], 2014. p. 16–23. Citado 3 vezes nas páginas 40, 42 e 44.

FARRIS, I. et al. Mifaas: A mobile-iot-federation-as-a-service model for dynamic cooperation of iot cloud providers. *Future Generation Computer Systems*, Elsevier, v. 70, p. 126–137, 2017. Citado 3 vezes nas páginas 39, 40 e 44.

GEISLER, K. The relationship between smart grids and smart cities. *IEEE Smart Grid Newsletter*, 2013. Citado na página 43.

GERLA, M. et al. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In: IEEE. *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. [S.l.], 2014. p. 241–246. Citado na página 14.

GIA, T. N. et al. Fog computing in healthcare internet of things: A case study on ecg feature extraction. In: IEEE. *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*. [S.l.], 2015. p. 356–363. Citado na página 25.

GIANG, N. K. et al. Developing iot applications in the fog: A distributed dataflow approach. In: IEEE. *Internet of Things (IOT), 2015 5th International Conference on the*. [S.l.], 2015. p. 155–162. Citado na página 49.

GRAFANA. <https://grafana.com/>, year=2018. Citado 3 vezes nas páginas 18, 58 e 59.

GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. Citado 3 vezes nas páginas 14, 16 e 20.

GUÉROUT, T.; ALAYA, M. B. Autonomic energy-aware tasks scheduling. In: IEEE. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*. [S.l.], 2013. p. 119–124. Citado 2 vezes nas páginas 17 e 46.

GUPTA, H. et al. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *arXiv preprint arXiv:1606.02007*, 2016. Citado 7 vezes nas páginas 15, 26, 39, 40, 42, 43 e 44.

HABAK, K. et al. Workload management for dynamic mobile device clusters in edge femtoclouds. In: ACM. *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. [S.l.], 2017. p. 6. Citado na página 38.

HASHEM, I. A. T. et al. The rise of "big data" on cloud computing: Review and open research issues. *Information systems*, Elsevier, v. 47, p. 98–115, 2015. Citado na página 22.

HE, J. et al. Multi-tier fog computing with large-scale iot data analytics for smart cities. *IEEE Internet of Things Journal*, IEEE, 2017. Citado 2 vezes nas páginas 40 e 44.

HE, X. et al. A novel load balancing strategy of software-defined cloud/fog networking in the internet of vehicles. *China Communications*, IEEE, v. 13, n. Supplement2, p. 140–149, 2016. Citado 3 vezes nas páginas 37, 39 e 42.

HODGES, S. et al. Prototyping connected devices for the internet of things. *Computer*, IEEE, v. 46, n. 2, p. 26–34, 2013. Citado na página 21.

HONG, K. et al. Mobile fog: A programming model for large-scale applications on the internet of things. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. [S.l.], 2013. p. 15–20. Citado na página 25.

HOQUE, S. et al. Towards container orchestration in fog computing infrastructures. In: IEEE. *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*. [S.l.], 2017. v. 2, p. 294–299. Citado 3 vezes nas páginas 40, 42 e 44.

HUANG, X.; GANAPATHY, S.; WOLF, T. Evaluating algorithms for composable service placement in computer networks. In: IEEE. *Communications, 2009. ICC'09. IEEE International Conference on*. [S.l.], 2009. p. 1–6. Citado na página 49.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, ACM, v. 40, n. 3, p. 7, 2008. Citado na página 28.

JARA, A.; LADID, L.; SKARMETA, A. The internet of things through ipv6: An analysis of challenges, solutions and opportunities. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, v. 3, n. 4, p. 97–118, 2014. Citado na página 20.

KALYVIANAKI, E. *Resource provisioning for virtualized server applications*. [S.l.], 2009. Citado 2 vezes nas páginas 26 e 27.

KAUR, K. et al. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE Wireless Communications*, IEEE, v. 24, n. 3, p. 48–56, 2017. Citado na página 36.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE, v. 36, n. 1, p. 41–50, 2003. Citado 2 vezes nas páginas 28 e 29.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 33.

KOPETZ, H. *Real-time systems: design principles for distributed embedded applications*. [S.l.]: Springer Science & Business Media, 2011. Citado na página 26.

KRAMER, J.; MAGEE, J. Self-managed systems: an architectural challenge. In: IEEE COMPUTER SOCIETY. *2007 Future of Software Engineering*. [S.l.], 2007. p. 259–268. Citado na página 28.

LEITNER, P. et al. Cost-efficient and application sla-aware client side request scheduling in an infrastructure-as-a-service cloud. In: IEEE. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.], 2012. p. 213–220. Citado na página 37.

LIGHT, R. A. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, The Open Journal, v. 2, n. 13, 2017. Citado na página 62.

LIU, M. et al. Performance analysis and optimization of handoff algorithms in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing*, IEEE, v. 7, n. 7, p. 846–857, 2008. Citado na página 21.

LIU, W.; SHINKUMA, R.; TAKAHASHI, T. Opportunistic resource sharing in mobile cloud computing: The single-copy case. In: IEEE. *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*. [S.l.], 2014. p. 1–6. Citado na página 38.

LIU, Y.; LEE, M. J.; ZHENG, Y. Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. *IEEE Transactions on Mobile Computing*, IEEE, v. 15, n. 10, p. 2398–2410, 2016. Citado 2 vezes nas páginas 38 e 39.

LOGHIN, D. et al. A performance study of big data on small nodes. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 8, n. 7, p. 762–773, 2015. Citado 2 vezes nas páginas 14 e 16.

LU, N. et al. Connected vehicles: Solutions and challenges. *IEEE internet of things journal*, IEEE, v. 1, n. 4, p. 289–299, 2014. Citado na página 26.

LUKE, S. Essentials of metaheuristics, 2013. *URL http://cs. gmu. edu/˜ sean/book/metaheuristics/Essentials. pdf*, 2017. Citado 2 vezes nas páginas 28 e 53.

MACHADO, J.; MORENO, E.; RIBEIRO, A. A review of computing fog and its research challenges. *Journal on Advances in Theoretical and Applied Informatics*, v. 3, n. 2, p. 32–39, 2017. ISSN 2447-5033. Disponível em: <http://revista.univem.edu.br/jadi/article/view/2469>. Citado 3 vezes nas páginas 15, 23 e 24.

MAHMUD, R.; KOCH, F. L.; BUYYA, R. Cloud-fog interoperability in iot-enabled healthcare solutions. In: ACM. *Proceedings of the 19th International Conference on Distributed Computing and Networking*. [S.l.], 2018. p. 32. Citado na página 40.

MAHMUD, R.; KOTAGIRI, R.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. In: *Internet of Everything*. [S.l.]: Springer, 2018. p. 103–130. Citado 3 vezes nas páginas 15, 42 e 44.

MASIP-BRUIN, X. et al. Managing resources continuity from the edge to the cloud: Architecture and performance. *Future Generation Computer Systems*, Elsevier, v. 79, p. 777–785, 2018. Citado na página 44.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. Citado 2 vezes nas páginas 22 e 23.

MELLO, R. de A. et al. An architecture for self-protection in internet of things. *ICWMC 2016*, p. 51, 2016. Citado na página 14.

MENNES, R. et al. Greco: A distributed genetic algorithm for reliable application placement in hybrid clouds. In: IEEE. *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. [S.l.], 2016. p. 14–20. Citado 7 vezes nas páginas 37, 38, 39, 40, 42, 44 e 45.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, Belltown Media, v. 2014, n. 239, p. 2, 2014. Citado na página 76.

MORABITO, R. et al. Evaluating performance of containerized iot services for clustered devices at the network edge. *IEEE Internet of Things Journal*, IEEE, v. 4, n. 4, p. 1019–1030, 2017. Citado 3 vezes nas páginas 40, 42 e 44.

MORELL, J. Á.; ALBA, E. Running genetic algorithms in the edge: A first analysis. In: SPRINGER. *Conference of the Spanish Association for Artificial Intelligence*. [S.l.], 2018. p. 251–261. Citado na página 38.

NALLUR, V.; BAHSOON, R. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Transactions on Software Engineering*, IEEE, v. 39, n. 5, p. 591–612, 2013. Citado na página 29.

NASSIFFE, R. et al. Optimising qos in adaptive real-time systems with energy constraint varying cpu frequency. *International Journal of Embedded Systems*, Inderscience Publishers (IEL), v. 8, n. 5-6, p. 368–379, 2016. Citado na página 37.

OPTIMIZER, I. C. {Available Online} http://www-01. ibm. com/software/commerce/optimization/cplex-optimizer. *Accessed May*, 2016. Citado na página 39.

OSANAIYE, O. et al. From cloud to fog computing: A review and a conceptual live vm migration framework. *IEEE Access*, IEEE, v. 5, p. 8284–8300, 2017. Citado na página 25.

PAHL, C.; LEE, B. Containers and clusters for edge cloud architectures–a technology review. In: IEEE. *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. [S.l.], 2015. p. 379–386. Citado na página 40.

PERALTA, G. et al. Fog computing based efficient iot scheme for the industry 4.0. In: IEEE. *Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017 IEEE International Workshop of*. [S.l.], 2017. p. 1–6. Citado na página 25.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, Elsevier, v. 64, p. 1–18, 2015. Citado 2 vezes nas páginas 31 e 33.

PLACHY, J.; BECVAR, Z.; STRINATI, E. C. Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In: IEEE. *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*. [S.l.], 2016. p. 1–6. Citado na página 36.

POORANIAN, Z. et al. A novel distributed fog-based networked architecture to preserve energy in fog data centers. In: IEEE. *Mobile Ad Hoc and Sensor Systems (MASS), 2017 IEEE 14th International Conference on*. [S.l.], 2017. p. 604–609. Citado 4 vezes nas páginas 39, 40, 42 e 44.

PORTAINER. <https://portainer.io/>. Citado na página 62.

PRINSLOO, T.; DEVENTER, J. P. V. Using the gartner hype cycle to evaluate the adoption of emerging technology trends in higher education–2013 to 2016. In: SPRINGER. *International Symposium on Emerging Technologies for Education*. [S.l.], 2017. p. 49–57. Citado na página 16.

PROMETHEUS. <https://prometheus.io/>, year=2018. Citado 3 vezes nas páginas 18, 58 e 61.

RACHKIDI, E. E. et al. Towards an efficient service provisioning in cloud of things (cot). In: IEEE. *Global Communications Conference (GLOBECOM), 2016 IEEE*. [S.l.], 2016. p. 1–6. Citado 5 vezes nas páginas 39, 40, 41, 42 e 44.

RENNER, T.; MELDAU, M.; KLIEM, A. Towards container-based resource management for the internet of things. In: IEEE. *Software Networking (ICSN), 2016 International Conference on*. [S.l.], 2016. p. 1–5. Citado 3 vezes nas páginas 40, 41 e 44.

RIVERA, J.; MEULEN, R. v. D. Gartner says the internet of things installed base will grow to 26 billion units by 2020. 2013. *URl: http://www. gartner. com/newsroom/id/2636073 (visited on 04/05/2016)*, 2014. Citado na página 14.

RODRIGUES, T. G. et al. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *IEEE Transactions on Computers*, IEEE, v. 66, n. 5, p. 810–819, 2017. Citado 2 vezes nas páginas 36 e 45.

ROMERO, F.; HACKER, T. J. Live migration of parallel applications with openvz. In: IEEE. *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*. [S.l.], 2011. p. 526–531. Citado na página 15.

SAHNI, S.; VARMA, V. A hybrid approach to live migration of virtual machines. In: IEEE. *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*. [S.l.], 2012. p. 1–5. Citado na página 15.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, ACM, v. 4, n. 2, p. 14, 2009. Citado na página 28.

SAMANIEGO, M.; DETERS, R. Hosting virtual iot resources on edge-hosts with blockchain. In: IEEE. *Computer and Information Technology (CIT), 2016 IEEE International Conference on*. [S.l.], 2016. p. 116–119. Citado 3 vezes nas páginas 40, 42 e 44.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The constrained application protocol (CoAP)*. [S.l.], 2014. Citado na página 63.

SHI, W. et al. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, IEEE, v. 3, n. 5, p. 637–646, 2016. Citado na página 16.

SHUKLA, A.; CHATURVEDI, S.; SIMMHAN, Y. Riotbench: An iot benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 29, n. 21, p. e4257, 2017. Citado na página 76.

SINGH, S.; CHANA, I. Earth: Energy-aware autonomic resource scheduling in cloud computing. *Journal of Intelligent & Fuzzy Systems*, IOS Press, v. 30, n. 3, p. 1581–1600, 2016. Citado 2 vezes nas páginas 17 e 46.

SINGH, S.; CHANA, I.; SINGH, M. The journey of qos-aware autonomic cloud computing. *IT Professional*, IEEE, v. 19, n. 2, p. 42–49, 2017. Citado 4 vezes nas páginas 17, 35, 36 e 46.

SKARLAT, O. et al. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, Springer, p. 1–17, 2017. Citado 7 vezes nas páginas 35, 38, 39, 40, 42, 44 e 48.

SKARLAT, O. et al. Towards qos-aware fog service placement. In: IEEE. *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*. [S.l.], 2017. p. 89–96. Citado 2 vezes nas páginas 37 e 45.

STERRITT, R. Autonomic computing. *Innovations in systems and software engineering*, Springer, v. 1, n. 1, p. 79–88, 2005. Citado na página 28.

STOJMENOVIC, I.; WEN, S. The fog computing paradigm: Scenarios and security issues. In: IEEE. *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. [S.l.], 2014. p. 1–8. Citado 2 vezes nas páginas 25 e 75.

SUÁREZ-ALBELA, M. et al. A practical evaluation of a high-security energy-efficient gateway for iot fog computing applications. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 9, p. 1978, 2017. Citado 2 vezes nas páginas 9 e 55.

SUNG, W.-T.; CHIANG, Y.-C. Improved particle swarm optimization algorithm for android medical care iot using modified parameters. *Journal of medical systems*, Springer, v. 36, n. 6, p. 3755–3763, 2012. Citado na página 14.

SUTO, K. et al. An energy-efficient and delay-aware wireless computing system for industrial wireless sensor networks. *IEEE Access*, IEEE, v. 3, p. 1026–1035, 2015. Citado 5 vezes nas páginas 37, 39, 41, 42 e 44.

TANEJA, M.; DAVY, A. Resource aware placement of iot application modules in fog-cloud computing paradigm. In: IEEE. *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. [S.l.], 2017. p. 1222–1228. Citado 7 vezes nas páginas 35, 37, 39, 40, 42, 44 e 45.

TANGANELLI, G.; VALLATI, C.; MINGOZZI, E. Energy-efficient qos-aware service allocation for the cloud of things. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*. [S.l.], 2014. p. 787–792. Citado na página 21.

TÄRNEBERG, W. et al. Dynamic application placement in the mobile cloud network. *Future Generation Computer Systems*, Elsevier, v. 70, p. 163–177, 2017. Citado 3 vezes nas páginas 36, 38 e 45.

TOCZÉ, K.; NADJM-TEHRANI, S. A taxonomy for management and optimization of multiple resources in edge computing. *Wireless Communications and Mobile Computing*, Hindawi, v. 2018, 2018. Citado na página 15.

URGAONKAR, R. et al. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation*, Elsevier, v. 91, p. 205–228, 2015. Citado 6 vezes nas páginas 36, 39, 40, 42, 44 e 45.

VELASQUEZ, K. et al. Service placement for latency reduction in the internet of things. *Annals of Telecommunications*, Springer, v. 72, n. 1-2, p. 105–115, 2017. Citado 5 vezes nas páginas 36, 38, 39, 44 e 45.

VERMA, S. et al. An efficient data replication and load balancing technique for fog computing environment. In: IEEE. *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*. [S.l.], 2016. p. 2888–2895. Citado 4 vezes nas páginas 39, 40, 42 e 44.

WALKER, M. J.; BURTON. Hype cycle for emerging technologies. *Gartner Inc*, p. 70, 2017. [Online; accessed 19-July-2018]. Citado na página 16.

WALKER, M. J.; BURTON, B.; CANTARA, M. Hype cycle for emerging technologies. *Gartner Inc*, p. 70, 2016. Citado na página 16.

WANG, S.; DEY, S. Cloud mobile gaming: Modeling and measuring user experience in mobile wireless networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, ACM, v. 16, n. 1, p. 10–21, 2012. Citado na página 25.

WANG, S. et al. Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 28, n. 4, p. 1002–1016, 2017. Citado 7 vezes nas páginas 37, 38, 39, 40, 42, 44 e 45.

WU, C.-M.; CHANG, R.-S.; CHAN, H.-Y. A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. *Future Generation Computer Systems*, Elsevier, v. 37, p. 141–147, 2014. Citado na página 56.

XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: IEEE. *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. [S.l.], 2013. p. 233–240. Citado na página 63.

XU, B. et al. Ubiquitous data accessing method in iot-based information system for emergency medical services. *IEEE Transactions on Industrial Informatics*, IEEE, v. 10, n. 2, p. 1578–1586, 2014. Citado 3 vezes nas páginas 14, 37 e 45.

XU, J. et al. Zenith: Utility-aware resource allocation for edge computing. In: IEEE. *Edge Computing (EDGE), 2017 IEEE International Conference on*. [S.l.], 2017. p. 47–54. Citado 4 vezes nas páginas 39, 40, 42 e 44.

YANG, C. Checkpoint and restoration of micro-service in docker containers. In: *Proc. 3rd Int. Conf. Mechatron. Ind. Informat*. [S.l.: s.n.], 2015. p. 915–918. Citado na página 76.

YI, S.; QIN, Z.; LI, Q. Security and privacy issues of fog computing: A survey. In: SPRINGER. *International conference on wireless algorithms, systems, and applications*. [S.l.], 2015. p. 685–695. Citado na página 75.

YOUSAF, F. Z.; TALEB, T. Fine-grained resource-aware virtual network function management for 5g carrier cloud. *IEEE Network*, IEEE, v. 30, n. 2, p. 110–115, 2016. Citado 2 vezes nas páginas 36 e 45.

ZAMANI, A. R. et al. Deadline constrained video analysis via in-transit computational environments. *IEEE Transactions on Services Computing*, IEEE, 2017. Citado na página 38.

ZENG, D. et al. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, IEEE, v. 65, n. 12, p. 3702–3712, 2016. Citado 3 vezes nas páginas 39, 40 e 42.

ZU, X.; BAI, Y.; YAO, X. Data-centric publish-subscribe approach for distributed complex event processing deployment in smart grid internet of things. In: IEEE. *Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on*. [S.l.], 2016. p. 710–713. Citado na página 51.

Anexos

ANEXO A – Artigos Aceitos e Publicados

Machine Learning Algorithms to Detect DDoS Attacks in SDN

Reneilson Santos[a,*], Danilo Souza[a], Walter Santo[a], Admilson Ribeiro[a], Edward Moreno[a]

[a] *Universidade Federal de Sergipe, São Cristóvão, Sergipe, Brazil*

Abstract

Software-Defined Networking (SDN) is an emerging network paradigm that has gained significant traction by many researchers to address the requirement of current data centers. Although central control is the major advantage of SDN, it is also a single point of failure if it is made unreachable by a Distributed Denial of Service (DDoS) attack. These attacks can swiftly incapacitate a victim, causing huge revenue losses. Despite the large number of traditional detection solutions that exist currently, DDoS attacks continue to grow in frequency, volume, and severity. This paper brings an analysis of the problem and suggests the implementation of some Machine Learning Algorithms (Support Vector Machine, Multilayer Perceptron, Decision Tree and Random Forest) with the purpose of classifying DDoS attacks in a SDN simulated environment (Mininet 2.2.2). With this goal, the DDoS attacks were simulated using the Scapy tool with a list of 20000 valid IPs, acquiring as a result the best accuracy with the Random Forest (99.19%) and a processing time of (21.8 ± 3.7)ms and the Decision Tree achieved the best processing time (10.0 ± 2.0)ms with the accuracy of 99.16%. In this way, the Decision Tree is considered more suitable to identify DDoS attacks in a SDN environment because of its lowest processing time and high accuracy.

Keywords: DDoS classification, Mininet, SDN environment, Random Forest, Decision Tree, SVM, MLP

1. Introduction

Many difficulties are found in current networks, such as: vertical integration, coupling between the data and controller plans, difficulty to modify (or insert) the application into the network, and decentralization, helped in the emergence of a new network paradigm would be able to eradicate these problems, the Software Defined Network, or SDN [1].

The great advantage of SDN is its capability to uncouple the data and controller plans, giving greater freedom to developers and facilitating the application maintenance in these environments [2].

To uncouple the control plane and let it logically centralized, it was possible to abstract the network infrastructure from its applications. Thus, services implemented through hardware (e.g., the IDS, Firewall, Routers, etc.) can be built using a software [3].

This centralization and uncouple of SDN is allowed because the network controller and the switches have well-defined programmable interfaces [1] through APIs, which allows the communication between them. The OpenFlow is one of the most remarkable API for the SDN environment [4], being one of the firsts APIs to be standardized.

The resources found in SDN allow the creation of components (applications) with low costs, if compared with the creation of these in current networks (the costs in SDN are only related with the time to program). Therefore, the SDN becomes an ideal environment to test new applications in a network.

Although the potential innovation introduced in the use of SDN, it brings some new networks security challenges [5]. These challenges can compromise basic properties of a secure communication network such as confidentiality, integrity, availability of information, authentication and non-repudiation [6].

In addition, DDoS attacks such as controller-switch communication flooding and switch flow-table flooding, may affect considerably a SDN environment due to the centralized controller and flow-table limitation in the network device, which can make critical services unavailable.

In IoT environments, devices as controllers, sensors and communication substrate, in critical infrastructures, can aggravate such problems [7]. Nonetheless, some solutions to SDN security challenges have already been proposed in the literature in the recent years [8, 9, 10, 11]. One technique that can be used to address these issues is the Machine Learning algorithms.

The popularity of the machine learning in recent years has seen the coupling of flow statistics with machine learning approaches to classify network traffic [12]. This approach is not new, despite its recent interest, a work published in 1994 [13] already addressed the intrusion detection.

The machine learning has significant advantages over static and payload inspection based techniques. The development of new approaches has been motivated by limitations in previous efforts. For instance, static classifiers are ineffective as they assume that applications use known port numbers that do not change.

In this context, this paper shows an IDS technique to detect some DDoS Attacks on a SDN environment with the implementation of different machine learning algorithms (simplified here by ML-Algorithms) (e.g., Multiple Layer Perceptron [MLP],

*Corresponding Author
Email address: reneilson1@gmail.com (Reneilson Santos)

Support Vector Machine [SVM], Decision Tree and Random Forest) and using the Scapy tool to generate the attacks and a Mininet with POX Controller to simulate the SDN environment.

In order to analyze the potential of the ML-Algorithms implementation on SDN (for the detection of DDoS attacks), this paper was divided as following: on Section 2 the related works are presented, followed by Section 3, that presents briefly the theoretical foundation of DDoS attacks and detection techniques; after that, Sections 4 and 5 present, respectively, the definition and execution of the experiment planning; concluding with Sections 6 and 7, that present the results obtained (with the validity threats) and the conclusion of this paper, respectively.

2. Related Works

In 2010, Braga et al. [14] made a DDoS Flooding Attack Detector using SOM (Self Organizing Maps), a machine learning unsupervised algorithm. They work with a NOX Controller, in which the switches of the network were monitored in predetermined periods. The detection works in three steps: the Flow Collector, the Feature Extractor and the Classifier (SOM). In the experiment, seven different types of flooding attacks were made using the Stacheldraht tool to generate the traffic. As a result, the SOM algorithm classifies the data in 271ms with 4-tuple and 352ms with 6-tuple (the tuple determines the number of features extracted), and, in terms of accuracy, in all cases, the Detection Rate measurement was more than 98%.

In 2014, Kokila et al. [11] propose an optimized protection scheme for SDN from flooding attacks based on the SVM classifier and they proposed a new algorithm called Idiletimeout Adjustment (IA). The authors analyzed the proposed scheme and evaluated the metrics in relation to the number of streams, the CPU usage of the SDN controller and of the OpenvSwitch.

Dao et al., in 2015, [15], created a method to track a type of DDoS attack that is worst in SDN than in normal networks (related to the time to process the first packet send by a host). The method, therefore, is characterized by a table and a count variable to obtain the IPs that are from real users and block those that are malicious. The experiment is simulated using an OPNET modeler and the result is less entries on flow table, less bandwidth and less packets on the controller. In addition to solve this problem, Mousavi and St-Hilaire in 2015 [16] created a method based on entropy, related to the destination IP address.

Tang et al. in 2016 [8] used MLP to detect intrusion in SDN, using six basic features easily obtained with Openflow functions. The MLP consists of three hidden layers, with the structure (12, 6, 3) and it is classified into two distinct classes, getting 75.75% of accuracy in the best case. The experiment was not performed in a real SDN environment, and, despite the architecture being given, nothing besides that regarding the simulation environment was given.

Nanda et al. in 2016 [17] used machine learning algorithms, trained on historical network attack data, to identify the potential malicious connections and potential attack destinations. It was used four widely-known machine learning algorithms: C4.5 (Decision Tree), Bayesian Network (BN), Decision Table (DT) and Naive-Bayes (NB) to predict the host that will be attacked based on the historical data. Experimental results show that the average prediction accuracy of 91.68% is attained using Bayesian Networks. Still on this paper, the authors used data from the LongTail project [18] to train the models. Leveraging the algorithm output, it is possible to define the security rules on the SDN controller to restrict the access of potential attackers by blocking the entire subnet.

Outside the SDN environment, many works could be found related to the use of classification algorithms to detect intrusion in networks. Dhanabal and Shantharajah [9], in 2015, carried out a study about some techniques and compared them, using the NSL-KDD as dataset, based on the KDD99 dataset, but with some improvements. The techniques used are J48 (a decision tree algorithm), SVM and Naive Bayes, and, for the DoS attack, the J48 was better than the other ones. Still regarding this matter, in 2015, Pervez and Farid [10] used the SVM with a different number of features and had good results (i.e. 99% of accuracy with all features of NSL-KDD).

3. DDoS Attacks in SDN

The purpose of this Section is to enlighten background information about the DDoS attacks in the SDN environment. First, we briefly describe DDoS attacks. This is followed by a description of the DDoS attack defensive mechanisms.

3.1. DDoS Attacks

Distributed Denial of Service (DDoS) attacks have been a real threat in many aspects of computer networks and distributed applications. The main objective of a DDoS attack is to bring down the services of a target using multiple sources that are distributed. For example, attackers can transfer thousands of packets to a victim to overwhelm its access bandwidth with illegitimate traffic, making online services unavailable. There are numerous denial of service (DoS) attack methods being used to degrade the performance, or availability of targeted services on the Internet [5]. Usually, these methods can be classified as challenges associated to the SDN at each layer of the framework, application, control and infrastructure.

3.1.1. Application Layer DDoS Attacks

Application layer attacks use softwares in a malicious way, aiming to exhaust resources to process any further requests. These attacks are generally harder to detect on the network level as they show no clear deviation from legitimate traffic [5].

Since the isolation of applications or resources in SDN is not well solved, DDoS attacks on one application can also affect other applications. A common example is the HTTP flooding attacks [19].

3.1.2. Control Layer DDoS Attacks

Controllers of SDN and their communications can be subjected to different types of attacks [2]. Among the threats that can cause significant damages are: attacks on the control plane and communication between the controller and other networks

components e.g, northbound API, southbound API, westbound API or eastbound API. In addition, the controller can be considered a single point of failure and scalability that raises potential performance problems and unavailability of the control plan.

3.1.3. Infrastructure Layer DDoS Attacks

Infrastructure layer DDoS attacks could potentially overload through two points: switches or by attacking the southbound API [20]. For example, a huge traffic may be sent by an attacker to execute a DoS attack on the node by setting up a number of new and unknown flows infrastructure layer.

3.2. DDoS Attack Detection Techniques

DDoS attacks have been studied for a long time and the types of threats to them are mostly known [5]. However, SDN is a new architecture and the studies are at an early stage. In addition, SDN networks have distinct detection methods for different types of DDoS attacks [21]. These methods include entropy-based [16, 22, 23, 24], machine learning-based [8, 9, 10, 11], traffic pattern analysis [25], connection rate [26, 25] and techniques that combine the use of the IDS and OpenFlow [27, 28, 29].

3.2.1. Entropy

The ability to measure randomness in packets arriving on a network makes entropy-based methods good candidates for the DDoS detection. The greater the randomness, the greater the entropy and vice versa. Entropy-based methods depend on network feature distributions to detect anomalous network activities [22]. The presence of anomalies in a SDN network can be identified by adopting the use of predefined thresholds. In addition, probability distributions of various network features such as source IP address, destination IP address, and port numbers are used to calculate the entropy [30].

3.2.2. Machine learning

Machine learning-based methods employ techniques to detect anomalies in a network environment, these can be based on models, based on statistic and math, based on unsupervised machine learning algorithms and based on supervised machine learning algorithms [31]. These algorithms take into account various network features and traffic characteristics to detect the presence of anomalies.

In fact, any system that is built to detect any anomalies in the network catch the traffic on it and extract some kind of information from them, and the approach that uses machine learning is trying to catch the pattern of normal and abnormal traffic on the network, without the need to know the pattern itself.

3.2.3. Traffic pattern analysis

These techniques work on the assumptions that the infected hosts exhibit similar behavioral patterns that are different from benign hosts [32]. Therefore, it analyzes the traffic relating to the metrics of the attack pattern networks in order to identify the attacker or the target under attack. Patterns are observed as a result of command that is sent to many members of the same botnet causing a similar behavior (e.g., sending illegitimate packets, starting to scan).

3.2.4. Connection rate

Bawany et. al [30] define connection rate techniques as "the probability of a connection attempt being successful should be much higher for a benign host than a malicious host". Whenever the likelihood ratio for a host to exceed a certain threshold, it is declared as infected. These techniques are classified into two types: connection success ratio and connection rate, that refers to the number of connections instantiated within a certain window of time.

3.2.5. SNORT and OpenFlow Integrated

Recent researches have combined the use of an intrusion detection system and OpenFlow to detect attacks and reconfigure the network dynamically [28, 30]. An intrusion detection system monitors the traffic to identify malicious activities. OpenFlow switches are then dynamically reconfigured based on the detected attacks in real time.

The information presented in the current Section gives an overview of the DDoS attacks and DDoS detection techniques in an SDN environment. In the next Section we present the definition and execution of the experiment planning.

4. Definition and Experiment Planning

In this Section, following the methodology described in [33], the experimental planning, related to the DDoS attack detection in the SDN environment, is presented.

The next subsections explain in details the definition of the experiment and its particularities, such as the goals and the planning of the experiment (that explains the context, shows the hypothesis, the project and what is used to achieve the goals and complete the experiment).

4.1. Goal Definition

The main goal of the experiment is to analyze ways to solve the problem of the DDoS attacks in the SDN environment. And a secondary goal is to compare different ML-Algorithms as a solution to the problem, analyzing metrics as accuracy and time to process in the SDN environment.

Following the formalization of the GQM model, proposed in [34], both objectives can be unified as: **To analyze** security methods to identify DDoS attacks in SDN, **with the purpose of** comparing them **with respect to** efficacy (accuracy) and efficiency (time to process) **from the point of view of** researchers and developers of SDN and Information Security **in the context of** a simulated DDoS attack on a Mininet Virtual Network with a POX Controller (Mininet VN).

4.2. Planning

Context Selection: the experiment was *in vitro*, in which the DDoS attack and normal traffic were simulated in the Mininet VN. In addition, to detect the anomalies and the normal traffic,

four different ML-Algorithms were used (MLP, SVM, Decision Tree and Random Forest).

Dependent Variables: accuracy and the time to process for each ML-Algorithm., besides their set of hyperparameters that could also be considered as dependent variables because each hyperparameter depends on the data used to train. Table 1 shows each hyperparameter used in this experiment associated with the respective machine learning model.

Table 1: Hyperparameters Table

Models	Hyperparameters
MLP	Activation Function
	Weight Update Function
	Hidden Layers
	L2 Regularization (α)
SVM	Kernel
	Regularization Factor (C)
	Classification Approach
Decision Tree (CART)	Selection Criteria
	Presort
	Maximum Number of Features
	Splitter
Random Forest	Number of Trees
	Selection Criteria
	Maximum Number of Features

Independent Variables: DDoS attacks and their parameters (presented on Table 2), the POX Component (implemented by the authors) and the algorithms by themselves.

Hypothesis Formulation: the research questions to this experiment are: could the problem with DDoS attacks be identified for future mitigation using a component in the SDN environment? Could some of the ML-Algorithms (MLP, SVM, Decision Tree or Random Forest) be considered better than all the others in terms of efficacy and efficiency?

According to these questions, the hypothesis that could be verified are presented below (to know, DT = Decision Tree and RF = Random Forest):

Hypothesis 1

H0: It is not possible to implement a component in SDN that can identify normal traffic and anomalies (specifically DDoS attacks) in a feasible way (time superior to 0.5s and the accuracy is less than 50%).

H1: There is a way to identify and classify the DDoS attacks and normal traffic using ML-Algorithms implemented as a component on the controller in the SDN environment (time inferior to 0.5s and accuracy superior to 50%).

Hypothesis 2

H0: There is no difference among the metrics of the algorithms $\mu_{SVM} = \mu_{MLP} = \mu_{DT} = \mu_{RF}$.

H1: Some algorithms have different metrics related to the others. ($\exists \mu_{alg_x} \neq \mu_{alg_y}$).

As mentioned before, the metrics used in this paper were the time to process (time to classify between anomaly or normal traffic) and accuracy.

Table 2: Features Description

Feature	Description
packet_count	Number of packets in flows
hard_timeout	Max time before discarding (seconds)
byte_count	Number of bytes in flow
duration_sec	Time flow has been alive in seconds
max_len	Max length to send to controller
type	Type of action
port	Output port
duration_nsec	Time flow has been alive in nanoseconds
priority	Priority level of flow entry
idle_timeout	Idle time before discarding (seconds)
cookie	Opaque controller-issued identifier
table_id	ID of the table to put the flow in
dl_type	Ethernet frame type
nw_dst	IP destination address
dl_src	Ethernet source address
nw_proto	IP protocol
nw_tos	Type of Service
tp_dst	TCP Destination Port
tp_src	TCP Source Port
dl_dst	Ethernet Destination Port
dl_vlan	Ethernet VLAN ID
nw_src	IP Source Port
in_port	Port ID

Objects Selection: the experiment used DDoS attacks simulated varying the IP source (list with more than 20000 IPs), port destination (choice depending on the application protocol or, when just a TCP packet or a UDP packet is sent, the port was chosen randomly), IP destination (6 hosts connected to the SDN environment), Transport Protocol (TCP and UDP) and Application Protocol (ICMP, HTTP, SMTP).

All the features are captured on the flow-table of the OpenFlow switch, simulated on the Mininet VN. Initially, some effort was made to choose the best set of features to classify the anomalies, however, the results showed that using all the features, without a long preprocessing step, the classification was expedited and did not significantly influence the results, therefore, it was chosen to maintain them all.

Experiment Project: every model used the same data (stored in a file) to evaluate the classification in terms of efficacy. In addition, all the DDoS attacks were generated randomly varying the independent variables, using the Scapy tool (`http://www.secdev.org/projects/scapy/`). Table 3 shows each traffic generated by each simulator.

Table 3: Traffic Simulators Table

Simulator	Attacks Used
Scapy	Normal Traffic
	ICMP Flooding
	HTTP Flooding
	TCP SYN Flood
	UDP Flood

In this way, we covered three different categories of vulnerabilities in the SDN environment:

- Controller Attack (Figure 1)
- Flow Table Attack (Figure 2)
- Bandwidth Between Switch and Controller Attack (Figure 3)

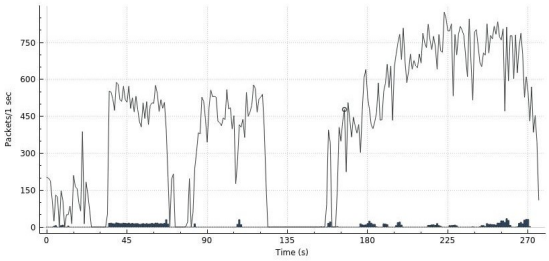


Figure 1: Controller Attack - Network Traffic

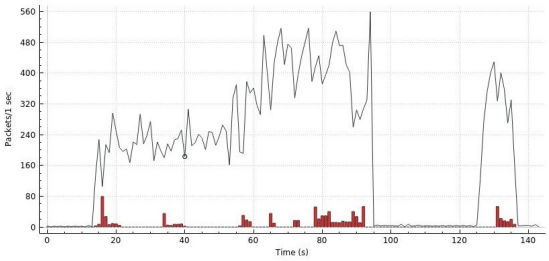


Figure 2: Flow Table Attack - Network Traffic

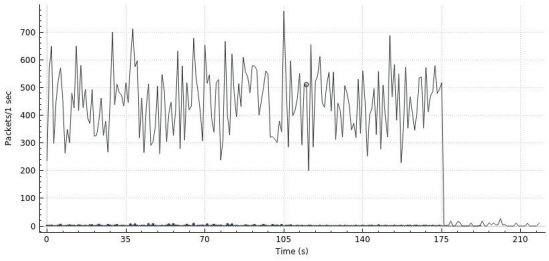


Figure 3: Bandwidth Attack - Network Traffic

Besides the normal traffic on the SDN environment (shown in Figure 4) The Controller Attack was made using the TCP SYN Flood, in which the Switch received a packet with a new Rule, sent to the Controller to create the rule for this packet. With a lot of new rules arriving at the Controller, it is overloaded and crashes. In this case, it was used 20000 different IPs, saved in a list and chosen randomly to avoid bias on the experiment.

The Flow Table Attack was made using HTTP flooding, ICMP flooding and UDP flooding, also using the same list of IPs used in the Controller Attack. In this case, each "client" was making, at most, ten requests to the servers (the hosts on

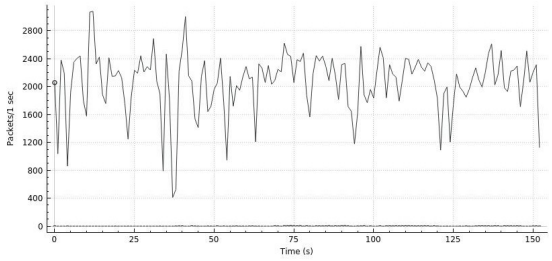


Figure 4: Normal Network Traffic

the Mininet VN), then, the table on the switch will increase until there is no more space to new rules, and, as a consequence, the time to respond to each new requisition increased.

Finally, the Bandwidth Attack was made using only the HTTP Flooding, in this case, the size of the IP packet was increased and sent to the hosts; the same list of IPs was used, but, in this case, the size of the packets was more important than the quantity of clients.

In all cases 6 different clients were used to send data simultaneously for the target hosts created on the Mininet environment.

In the next Section, an image of the SDN architecture for this experiment is shown in Figure 5.

Besides that, the models are implemented directly on the Mininet VN to get the efficiency metric (time to process) of each model.

Instrumentation: the algorithms were obtained through Python libraries at the 3.5.1 version. The execution of the algorithms to get the best hyperparameters was in a HP desktop, with 4Gb of RAM, Intel i5-3470S (2.9GHz) executing in an Ubuntu 12.04.5 LTS operational system. The acquisition of the metrics to verify the hypothesis was made in an Virtual Machine, with 1Gb of RAM, Intel i5-3470S (2.9GHz) executing in an Mininet 2.2.2 on Ubuntu 14.04 LTS - 32 bit, in which the controller is the POX.

5. Experiment Execution

5.1. Preparation

According to [21], the DDoS attacks on the SDN environment have new challenges to deal with (attack on the Controller, attack on the dataflow table and attack on the bandwidth between the switch and the controller are some examples), in addition to the problems already known about this type of attack on common network environments. Despite these new challenges in SDN, this new technology also allows new solutions, using the dataflow captured from the switch and deal with it in the controller (implementing a new solution in software).

Therefore, in this paper the Scapy tool is used to generate the traffic flow (normal and malicious), using different protocols, different IP sources, different port numbers and some others variables randomly chosen. It generated the three examples of new challenges related to DDoS attacks on the SDN environment (Attack on the Controller, Attack on the Dataflow Table

and Attack on the Bandwidth), already detailed on the last Section.

The experiment is performed using Mininet VN by contextualizing a Smart Home IoT network, a similar environment to the one shown in Figure 5, with six hosts (simulating the IoT components connected to the SDN), one switch and one controller.

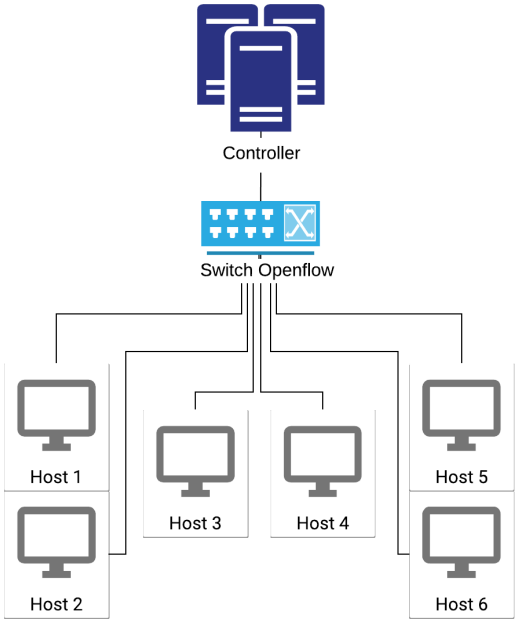


Figure 5: Mininet Architecture Illustration

After the construction of the environment in the Mininet VN, the next step is to implement a POX component to capture the dataflow on the switch OpenFlow and store in a file to be used by the algorithms as train and test data.

Finally, the final step was implement the component to detect the DDoS attack with the ML-Algorithms already trained on the previous step and observe its behavior on the simulated SDN environment.

5.2. Data Collection

To collect the data, a specific POX component was codified in the SDN environment to store in a file the dataflow.

Firstly, in six clients, the Scapy tool was used to generate malicious data traffic.

Secondly, to obtain traffic in the network, using six clients again, the "ping" command and the Scapy tool were used, generating HTTP and ICMP normal traffic.

In both cases (malicious and normal traffic), the six hosts of the simulated environment were the target.

At the end, 70% of the dataset (containing 20000 data of the dataflow table, being 10000 for each type of traffic) was used to evaluate the hyperparameters of the models. The *GridSearch* technique, implemented by the *Scikit-learn* [35], a Python Machine Learning library, allowed the local search of the best hyperparameters set. In order to avoid model overfitting (specially for SVM and MLP models), standardization of the features values was applied on the data.

It is important to remember that, the 10000 data used as malicious were captured from three different kind of attacks (3333 as Attack on Controller, 3333 as Attack on Dataflow Table and 3334 as Attack on Bandwidth).

Once the best hyperparameters set for each model were achieved, the accuracy was obtained using the test dataset (30% of the dataset randomly chosen).

After that, to get the efficiency metric, the models were implemented in the Mininet VN and the data was obtained by the dataflow table in each 2s and processed by the model, calculating the differential time between the start and the end of the execution of the classification. In this part of the experiment, the data (time to process) were saved in a file (a total of 50 different acquisitions) to be analyzed *a posteriori*, using statistical analysis.

Moreover, to test the accuracy for each kind of attack, three others dataset were created to achieve this efficacy metric, each one with 10000 of normal attack and 10000 of some other attack (controller, bandwidth and flow table), in this way, it is possible to analyze the efficacy of the ML-Algorithms for each kind of DDoS attack.

Finally, all the data acquired were validated as shown in the following subsection.

5.3. Data Validation

In the training step, the algorithm used the K-fold technique [36], which avoid overfitting of the algorithm.

Related to the efficiency metric, the Kolmogorov-Smirnov test (KS-Test) was used to analyze if the data follow a normal distribution, what is refuted by a low p-value (close to zero), indicating that the time to process does not follow a normal distribution for all models (Random Forest, Decision Tree, SVM and MLP).

After that, these data were analyzed using the Friedman Test to verify the hypothesis raised and the result was that there are statistical differences in the mean of the time to process of the algorithms, just the MLP and the Decision Tree have no statistical difference according to the result of the test.

In the next Section, the results and a discussion about them are presented in more details.

6. Results

Table 4 shows the best hyperparameters set for each ML-Algorithm.

The Figure 6 shows the total accuracy (given by the Equation 1) of the ML-Algorithms for each attack simulated.

$$Accuracy = \frac{Number\ of\ Correct\ Classifications}{Total\ of\ Samples} \quad (1)$$

The accuracy is a statistical value that determines how close our ML-Algorithm is to the ideal, if this value is 1, it means that the algorithm has no error and classifies the data perfectly. This is a good metric when the data is equally distributed (the same

Table 4: hyperparameters Table

Models	hyperparameters
MLP	Activation Function: Relu Weight Update Function: Quasi-Newton Method (*lbfgs*) Layers: 1 hidden (15 neurons) L2 Regularization (α) : 1e-3
SVM	Kernel: RBF Regularization Factor (C) : 1e+5 Classification Approach : One against rest (*ovr*)
Decision Tree (CART)	Selection Criteria : Gini Presort : True Maximum Number of Features : $\sqrt{Features}$ Splitter : best
Random Forest	Number of Trees : 25 Selection Criteria : Entropy Maximum Number of Features: $\sqrt{Features}$

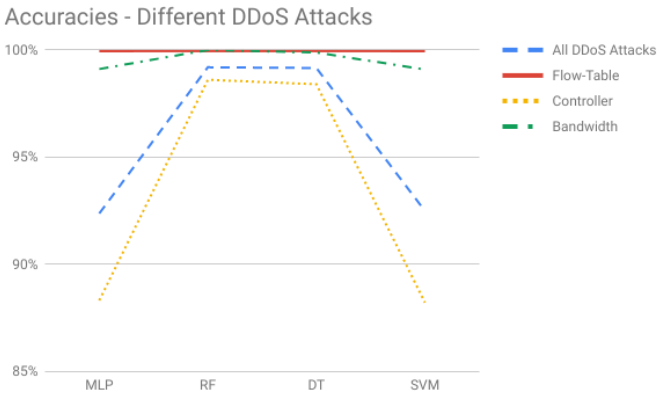


Figure 6: Accuracies of ML-Algorithms for Each DDoS Attack

quantity of data for each class), as is the case of our experiment [36].

Figure 7 shows the ROC curve for the classification of all the three kinds of attack simulated.

In addition, Figure 8, Figure 9 and Figure 10 show the ROC curve for each kind of DDoS attack simulated in this experiment (attack on controller, attack on the flow-table and attack on the bandwidth between the controller and switch overflow, respectively).

The ROC curve represents a relation between sensitivity (in our experiment represented by the percentage of data classified as malicious that is really malicious) and $1 - specificity$ (in our experiment, the specificity is represented by the percentage of data classified as malicious that is really malicious, then 1 minus this value is the opposite, i.e., the percentage of data classified as normal but that are malicious) [36].

This curve is very used in the Machine Learning to choose a good point for the classifiers, given by the point above the central curve in which the distance between them is maximum.

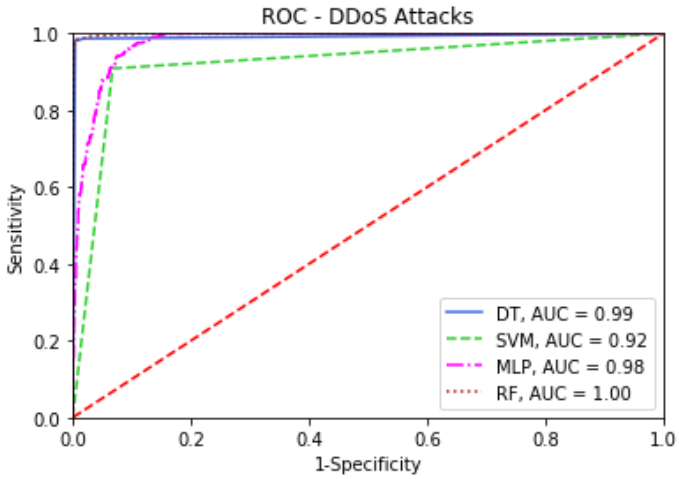


Figure 7: ROC Curve - All DDoS Attacks

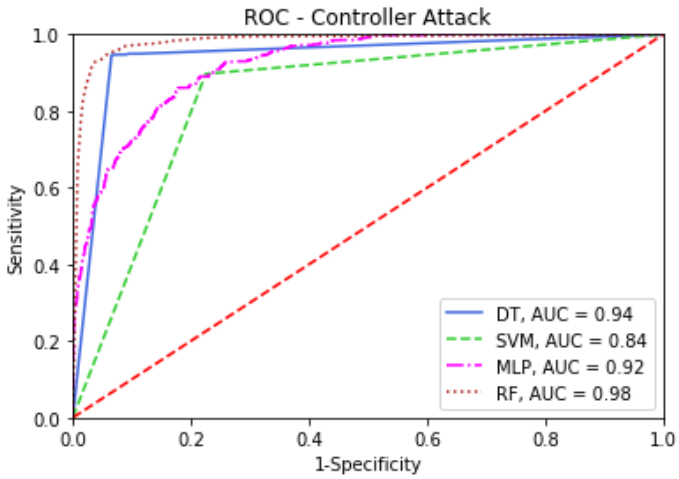


Figure 8: ROC Curve - Controller Attack

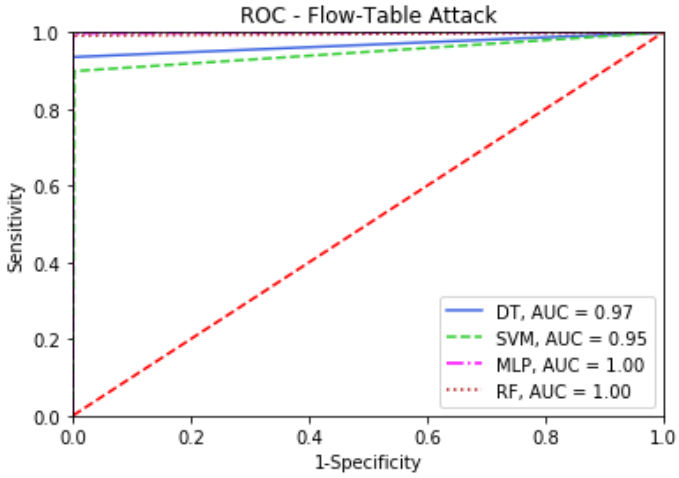


Figure 9: ROC Curve - Flow-Table Attack

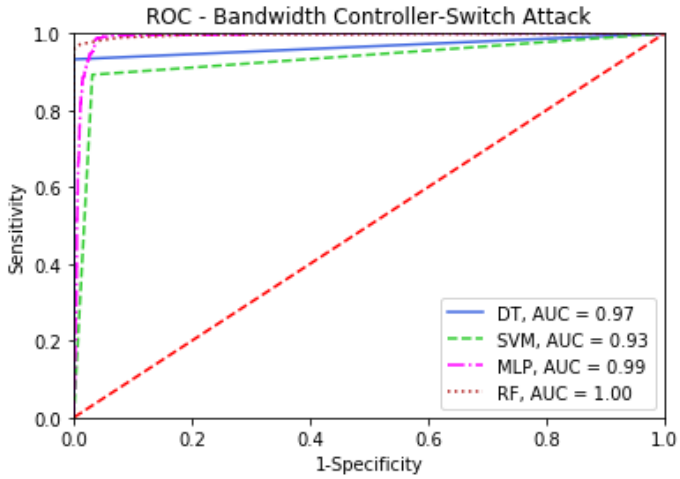


Figure 10: Bandwidth Controller-Switch Attack

In the graphs, it shown the AUC metric as well, that is the Area Under the Curve, the higher this metric, the best is the classification, therefore, that is a good metric to evaluate any ML-Algorithm that classify a binary system (as is the case of our experiment, malicious vs normal traffic).

Finally, as the result of the time to process each algorithm, Figure 11 shows algorithm behavior in the classification of DDoS simulated attacks on the SDN.

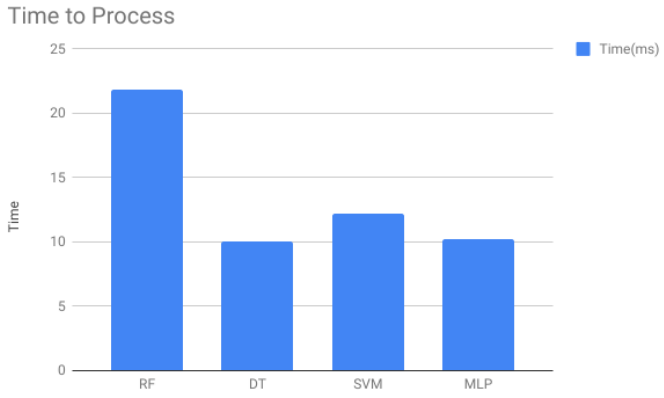


Figure 11: Time To Process

6.1. Analysis and Interpretation

As explained in the previous Section, the algorithms were trained using the K-fold technique [36], in this way, it is avoided overfitting of the classifiers. Regarding that, the results shown in Figure 6 is related to the 30% of the data separated for test, in other words, 70% of the data were used as training data, in which the K-fold technique was applied with K=10 and the others 30% of the data were used to test the algorithms, what confirms if the algorithms were not overfitted.

As a result for all kinds of attack, both the Random Forest and the Decision Tree have good results in terms of efficacy, with almost 100% of accuracy. However, as we can see in Figure 11, the Decision Tree had the lowest processing time while the Random Forest had the largest one, knowing that, in this context of a DDoS attack, it is more important take in consideration the time to process the algorithm and, as a consequence, it is possible to say that the more suitable algorithm for this problem is the Decision Tree.

Moreover, related to the attacks analyzed separately, it is possible to notice that the Flow-Table Attack is the one with the best metrics of efficacy in the process of classification, while the Controller Attack has the worst. In addition, it is possible observe that the Random Forest has always the best result in all three cases.

The problem with these algorithms, however, is when a new type of attack happens to the network, they will not capture them. In other words, the implemented algorithms were not online, in which new information obtained and identified as a DDoS attack by a specialist improve the model. This type of classifier is intended to be implemented in a future work, even because as explained in Section 4, the main goal of this paper was to prove that it is possible to identify the new kinds of DDoS attack in the SDN environment using algorithms already known in the literature, which is confirmed by the good results achieved.

As a conclusion, we can say that using a Decision Tree algorithm in a SDN environment improve the security without big losses on the traffic of the network because this is a ”light” algorithm, with no high processing power or memory consumption, ideal for the network environment.

6.2. Threats to Validity

A K-fold algorithm was used as a way to mitigate biases related to concluding remarks in regards to the established hypothesis (**conclusion validity**). Besides that, still related to conclusion validity and **external validity**, a database with attacks randomly generated and with a big list of IP sources were used.

A particular threat to validity is the fact that all traffic the have been generated artificially (**conclusion validity**), however, as a way to be near to the real world network, some variables have been randomly generated (or chosen), as IP, protocols, size of packets and quantities of requisitions by a unique ”client”.

7. Conclusion

With the emergence of SDN environments, in which the possibility of make our own applications to mitigate security issues, or propose new software solutions on the network environment, have made many researchers and industries look in a good way for this new concept.

However, together with the new possibilities to build solutions, SDN also brings some new problems to be treated. One of these problems is new kinds of DDoS attacks. For example, the DDoS attack in the SDN environment could attack the centralized controller and put down all the network if the controller crashes.

In this paper was proposed the use of ML-Algorithms (MLP, SVM, Decision Tree and Random Forest) to detect DDoS attacks in three different categories (attack to the flow table, attack to the bandwidth and attack to the controller). Every attack was made using the Scapy tool, using a list with more than 20000 IPs used as attackers, and the hosts connected to the SDN network as targets, simulated using the Mininet VN.

As a result of the implementation of the ML-Algorithms as detectors of these types of attacks, the Decision Tree was found to be the best in general, mainly because it has the lowest time to process (although the Random Forest have has the best accuracy in absolute value).

As a conclusion, in future works we intend to use real traffic and implement a technique to mitigate the DDoS attacks or avoid them as soon as possible. In addiction, a new experiment in a prototyped SDN environment using Raspberry Pi platform is also an intended future work, which will provide more support to the proposed solution.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[2] B. Wang, Y. Zheng, W. Lou, Y. T. Hou, Ddos attack protection in the era of cloud computing and software-defined networking, Computer Networks 81 (2015) 308–319.

[3] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for sdn? implementation challenges for software-defined networks, IEEE Communications Magazine 51 (7) (2013) 36–43.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[5] S. Scott-Hayward, G. O'Callaghan, S. Sezer, Sdn security: A survey, in: Future Networks and Services (SDN4FNS), 2013 IEEE SDN For, IEEE, 2013, pp. 1–7.

[6] C. Douligeris, D. N. Serpanos, Network security: current status and future directions, John Wiley & Sons, 2007.

[7] M. Özçelik, N. Chalabianloo, G. Gür, Software-defined edge defense against iot-based ddos, in: Computer and Information Technology (CIT), 2017 IEEE International Conference on, IEEE, 2017, pp. 308–313.

[8] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on, IEEE, 2016, pp. 258–263.

[9] L. Dhanabal, S. Shantharajah, A study on nsl-kdd dataset for intrusion detection system based on classification algorithms, International Journal of Advanced Research in Computer and Communication Engineering 4 (6) (2015) 446–452.

[10] M. S. Pervez, D. M. Farid, Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms, in: The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014), 2014, pp. 1–6. doi:10.1109/SKIMA.2014.7083539.

[11] R. Kokila, S. T. Selvi, K. Govindarajan, Ddos detection and analysis in sdn-based environment using support vector machine classifier, in: Advanced Computing (ICoAC), 2014 Sixth International Conference on, IEEE, 2014, pp. 205–210.

[12] J. Zhang, X. Chen, Y. Xiang, W. Zhou, J. Wu, Robust network traffic classification, IEEE/ACM Transactions on Networking (TON) 23 (4) (2015) 1257–1270.

[13] J. Frank, Artificial intelligence and intrusion detection: Current and future directions, in: Proceedings of the 17th national computer security conference, Vol. 10, Baltimore, MD, 1994, pp. 1–12.

[14] R. Braga, E. Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: Local Computer Networks (LCN), 2010 IEEE 35th Conference on, IEEE, 2010, pp. 408–415.

[15] N.-N. Dao, J. Park, M. Park, S. Cho, A feasible method to combat against ddos attack in sdn network, in: Information Networking (ICOIN), 2015 International Conference on, IEEE, 2015, pp. 309–311.

[16] S. M. Mousavi, M. St-Hilaire, Early detection of ddos attacks against sdn controllers, in: Computing, Networking and Communications (ICNC), 2015 International Conference on, IEEE, 2015, pp. 77–81.

[17] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, B. Yang, Predicting network attack patterns in sdn using machine learning approach, in: Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on, IEEE, 2016, pp. 167–172.

[18] LongTail, Longtail log analysis., [Online; access 08/31/2018]. URL http://longtail.it.marist.edu/honey/

[19] C. Wang, T. T. Miu, X. Luo, J. Wang, Skyshield: A sketch-based defense system against application layer ddos attacks, IEEE Transactions on Information Forensics and Security 13 (3) (2018) 559–573.

[20] Q. Yan, F. R. Yu, Distributed denial of service attacks in software-defined networking with cloud computing, IEEE Communications Magazine 53 (4) (2015) 52–59.

[21] Q. Yan, F. R. Yu, Q. Gong, J. Li, Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges, IEEE Communications Surveys & Tutorials 18 (1) (2016) 602–622.

[22] R. Wang, Z. Jia, L. Ju, An entropy-based distributed ddos detection mechanism in software-defined networking, in: Trustcom/BigDataSE/ISPA, 2015 IEEE, Vol. 1, IEEE, 2015, pp. 310–317.

[23] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, V. Maglaris, Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments, Computer Networks 62 (2014) 122–136.

[24] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, A. Schaeffer-Filho, Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn, in: Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP, IEEE, 2016, pp. 27–35.

[25] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, M. Tyson, Fresco: Modular composable security services for software-defined networks, in: 20th Annual Network & Distributed System Security Symposium, NDSS, 2013.

[26] S. Dotcenko, A. Vladyko, I. Letenko, A fuzzy logic-based information security management for software-defined networks, in: Advanced Communication Technology (ICACT), 2014 16th International Conference on, IEEE, 2014, pp. 167–171.

[27] L. Hsiao-Chung, W. Ping, Implementation of an sdn-based security defense mechanism against ddos attacks, DEStech Transactions on Economics, Business and Management (iceme-ebm).

[28] J. S. Martins, M. B. Campos, A security architecture proposal for detection and response to threats in sdn networks, in: ANDESCON, 2016 IEEE, IEEE, 2016, pp. 1–4.

[29] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, H. Lim, Suspicious traffic sampling for intrusion detection in software-defined networks, Computer Networks 109 (2016) 172–182.

[30] N. Z. Bawany, J. A. Shamsi, K. Salah, Ddos attack detection and mitigation using sdn: methods, practices, and solutions, Arabian Journal for Science and Engineering 42 (2) (2017) 425–441.

[31] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, J. Srivastava, A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection, pp. 25–36. arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972733.3, doi:10.1137/1.9781611972733.3.
URL https://epubs.siam.org/doi/abs/10.1137/1.

9781611972733.3
[32] R. Harang, Bridging the semantic gap: Human factors in anomaly-based intrusion detection systems, in: Network Science and Cybersecurity, Springer, 2014, pp. 15–37.
[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.
[34] V. R. Basili, G. Caldiera, H. D. Rombach, The goal question metric approach, Encyclopedia of software engineering 2 (1994) (1994) 528–532.
[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[36] R. S. Michalski, J. G. Carbonell, T. M. Mitchell, Machine learning: An artificial intelligence approach, Springer Science & Business Media, 2013.

Avaliação de Performance para Fornecer StaaS a Dispositivos IoT em Ambiente Fog Computing

José dos Santos Machado, Danilo Souza Silva, Raphael Silva Fontes, Adauto Cavalcante Menezes, Edward David Moreno, Admilson de Ribamar Lima Ribeiro

Departamento de Computação, Universidade Federal de Sergipe – UFS – Brasil

jsmac18@hotmail.com, {danilosilva.se, raphaelf.ti, adauto.cavalcant, edwdavid}@gmail.com, admilson@ufs.br

Resumo. *Este trabalho apresenta a Fog Computing, sua contextualização teórica, os trabalhos correlatos e realiza avaliação de uma Fog Computing, para fornecer StaaS (Storage as a Service), a dispositivos IoT utilizando plataformas de sistemas embarcados e compara seus resultados com os obtidos por um servidor de alto desempenho. Os resultados demonstraram que a implementação desse serviço em dispositivos de sistemas embarcados pode ser uma boa alternativa para reduzir um desses problemas, no caso, o armazenamento de dados, que atinge hoje os dispositivos IoT.*

1. Introdução

Nas últimas décadas, observa-se que serviços de computação como armazenamento, processamento de dados e controle foram transferidos para a "nuvem". A oportunidade de computação ilimitada na nuvem permite que os usuários finais acessem amplas informações facilmente, também é possível visualizar que os dispositivos móveis e sensores, como *smartphones*, se tornaram poderosos equipamentos, o que levou ao surgimento de novos sistemas e aplicações.

Estes sistemas e aplicações introduzem novas demandas funcionais em computação e redes que a "nuvem" sozinha não pode atender. Neste cenário percebe-se que a computação local na borda da rede é muitas vezes necessária [7]. Por exemplo, para processar dados em tempo real, criar contextos de reconhecimento de localização a partir de sensores locais e maximizar a eficiência das comunicações sem fio na borda da rede. Em geral, a nuvem está muito longe dos dispositivos para satisfazer requisitos de latência e, é muito centralizada para lidar com a heterogeneidade e diversidade contextual em uma área local [14].

Para ultrapassar estas limitações, porções da capacidade de computação da nuvem podem ser deslocados para a borda da rede e formam um ambiente de computação local, isto é, uma "*Fog Computing*" chamada também de "nevoeiro" [7]. Ao distribuir os serviços de computação e de rede mais próximos de onde os dados do usuário são gerados, a *Fog* atende melhor às demandas emergentes [13].

A *Fog Computing* apresenta uma nova arquitetura que "leva processamento para os dados", enquanto a nuvem "leva os dados para o processamento" [1]. Dessa maneira dispositivos de borda e dispositivos móveis podem estar interligados dentro de uma rede local e executar colaborativamente tarefas de armazenamento, processamento de dados de rede e de controle [4].

A *Fog Computing* pode vir a resolver muitos problemas da Internet das Coisas (IoT), por exemplo, os serviços da *Fog* serão capazes de melhorar a largura de banda e as restrições de custo das comunicações de longo alcance. No entanto, muitos desafios ainda permanecem na computação em *Fog*, como modelar uma arquitetura de sistema para a *Fog*; como implementar, organizar e gerenciar dispositivos da *Fog*; como a *Fog* interage com a nuvem e com os dispositivos; e como gerenciar a conectividade física e lógica da *Fog*; entre outros.

Este trabalho apresenta o conceito da *Fog Computing*, os trabalhos correlatos e realiza a análise do fornecimento StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataformas de sistemas embarcados em um ambiente *Fog Computing* e compara seus resultados com os obtidos por um servidor de alto desempenho.

O artigo está organizado em seis seções, a seção 2 apresenta o conceito e características da *Fog Computing*, na seção 3 é dedicada a revisão da literatura, a seção 4 temos o cenário de teste, na seção 5 avaliação e resultados e por fim na seção 6 as conclusões e trabalhos futuros.

2. Fog Computing

Devido à latência de rede frequentemente imprevisível, especialmente em um ambiente móvel, muitas vezes a computação em nuvem não pode atender aos requisitos rigorosos de latência, segurança e privacidade dos aplicativos em área restrita geograficamente [16]. Por outro lado, a crescente quantidade de dados gerados por dispositivos e sistemas, com poucos recursos pode se tornar impraticável para transportar dados através de redes para nuvens remotas [11].

Para isso, surgiu um novo paradigma, a *Fog Computing*. O conceito de computação em *Fog* foi adotado pela *Cisco Systems* como um novo paradigma em 2012 [4]. A *Fog Computing* é a computação em nuvem que distribuirá serviços avançados de computação, armazenamento, rede e gerenciamento mais próximos dos usuários finais, enviando informações dos dispositivos IoT para *Cloud Computing*, formando assim uma plataforma distribuída e virtualizada, assim, também é referido como computação de borda [6].

2.1 Características da Fog Computing

A computação *Fog* é um paradigma inovador que realiza computação distribuída, serviços de rede e armazenamento, além da comunicação entre *Cloud Computing Data Centers* até os dispositivos ao longo da borda da rede. Essa comunicação amplia as operações e serviços inerentes à computação em nuvem, permitindo assim uma nova geração de aplicativos [11]. A principal função é filtrar e agregar dados para os centros de dados da *Cloud* e aplicar inteligência lógica a dispositivos finais [7]. A figura 1 mostra a localização entre *Fog Computing* e *Cloud Computing*. A figura 2 apresenta arquitetura da *Fog Computing* com a sua localização.

Fig. 1 - Fog Localizada Entre Borda e Nuvem [14]

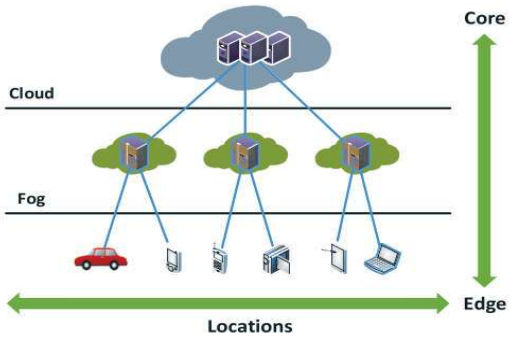
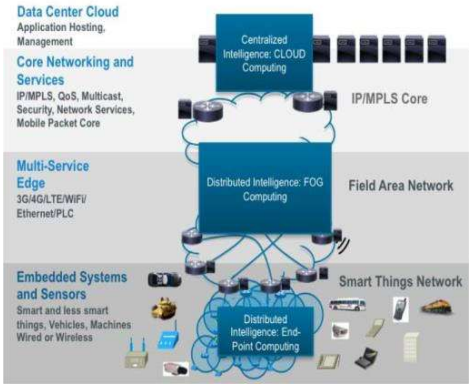


Fig. 2 - Arquitetura da Fog Computing [16]



Fog Computing é semelhante à computação em nuvem em muitos aspectos, no entanto, pode ser diferenciado do anterior pelo fato de estar próximo dos dispositivos finais, a distribuição espacial geograficamente menor e a possibilidade de apoio à mobilidade [7]. Como o processamento baseado em *Fog* ocorre ao longo da borda da rede, os resultados finais refletem uma percepção de localização altamente melhorada, baixa latência e Qualidade de Serviço (QoS), em aplicações de streaming e tempo real, os nós de nevoeiro são dispositivos heterogêneos, que vão desde servidores, pontos de acesso, roteadores de borda até dispositivos finais como telefones celulares, relógios inteligentes, sensores e etc [1].

2.2 Armazenamento como Serviço (StaaS)

A computação em nuvem e, em particular, os serviços de armazenamento em nuvem tornaram-se uma parte cada vez mais importante do setor de tecnologia da informação nos últimos tempos. O número de opções de armazenamento em nuvem está aumentando, com a maioria dos fornecedores fornecendo uma quantidade variada de armazenamento livre antes de cobrar por níveis de armazenamento maiores, devido ao número crescente desses serviços disponíveis, muitos pesquisadores usaram a frase *Storage as a Service* (StaaS), como uma extensão do *Software as a Service*, para descrever esse tipo de serviço [9].

3. Trabalhos Analisados

Um total de oito artigos foram identificados na implementação da *Fog Computing*, para melhorar alguns dos serviços da integração entre a *Cloud* e os dispositivos IoT, apresentaremos seus resultados e seus respectivos trabalhos futuros de pesquisas.

ZHU *et al*. (2013) [16], apresentaram a otimização da web dentro do contexto *Fog Computing*. Aplicaram métodos existentes para otimização da web de uma maneira inovadora, de tal forma que, esses métodos podem ser combinados com conhecimento exclusivo que está disponível apenas nos nós de borda (*Fog*). Como trabalho futuro sinalizaram aplicar seus conceitos propostos para desenvolver um sistema de prova de conceito.

No trabalho de CRACIUNESCU *et al.* (2015) [2], apresentaram uma implementação em laboratório de e-Saúde, onde o processamento em tempo real é realizado pelo PC doméstico, enquanto os metadados extraídos são enviados para a nuvem para processamento posterior. Como trabalho futuro sinalizaram adicionar mais sensores e mais dispositivos *off-the-shelf*, que são atualmente *mainstream*, e os dados de fusão desses dispositivos.

Em [15] VERBA *et al*. (2016), analisaram as plataformas existentes e suas deficiências, bem como propuseram uma plataforma de gateway modular, baseada em mensagens que permite o agrupamento de *gateways* e a abstração dos detalhes do protocolo de comunicação periférica. E sinalizaram como trabalho futuro desenvolver um ambiente de laboratório inteligente com diversos dispositivos mais complexos e cenários de controle para testar completamente o sistema.

FAN *et al*. (2016) [3], apresentaram a capacidade do recurso de *Web Caching* adicionado ao dispositivo de borda para servir como servidor *proxy* de armazenamento em cache, para obter mais armazenamento em cache. Os dispositivos finais também são explorados para fornecer algum espaço de cache. Como trabalho futuro sinalizaram adicionar mais funcionalidades ao dispositivo de borda, como a segurança e implementar o trabalho no mundo real.

No trabalho de HAO *et al.* (2017) [5], apresentaram um design do WM-FOG, uma estrutura de computação para ambientes *Fog*, que engloba essa arquitetura de *software* e avalia seu protótipo do sistema. Como trabalho futuro sinalizaram adicionar mais recursos ao WM-FOG para melhor atender às aplicações de computação em *Fog*.

Em [8] LI *et al.* (2017), discutiram dois conceitos de codificações recentemente propostos, códigos de mínima largura de banda e códigos de mínima latência, e ilustram seus impactos na computação em *Fog*, também analisaram uma estrutura de codificação unificada que inclui as duas técnicas de codificação acima descritas. Como trabalho futuro sinalizaram a necessidade de pesquisar computação heterogênea; redes com topologia de camada múltipla e estruturada; tarefas de computação em várias etapas; custos de computação codificados; verificar a computação distribuída; explorar as estruturas algébricas das tarefas computacionais; aplicações pesadas de comunicação e nós de *Fog plug-and-play*.

OSANAIYE *et al.* (2017) [11], apresentaram uma abordagem conceitual de migração em tempo real de pré cópia para a migração de VM e sinalizou como trabalho futuro a implantação do *framework* no mundo real ou ambiente de teste, com o objetivo de sua validação.

E por fim, POPENTIU-VLADICESCU et al. (2017) [12], analisaram os modelos de arquiteturas e práticas existentes em *Fog Computing* visando a confiabilidade e segurança dos sistemas de *Fog*, uma abordagem considerada integradora de três componentes da confiabilidade do sistema: a confiabilidade dos nós, a confiabilidade da rede e a confiabilidade do *software*, a arquitetura de referência *OpenFog* e os esquemas AJIA e BDSC. Como trabalho futuro sinalizaram resolver problemas técnicos e de algoritmos no paradigma de *Fog Computing*.

No quadro 1 os dados dos artigos analisados são sumarizados e comparados com este trabalho, foram organizados na ordem crescente cronologicamente para demostrar a

evolução em relação ao tema *Fog Computing*. Os artigos foram comparados quanto a implementação da *Fog Computing*, a utilização de plataforma de *Cloud*, utilização de dispositivo embarcado, uso de técnicas ou métodos de avaliação (*Benchmark*) e a implementação do StaaS.

Tabela 1 - Comparação Entre os Trabalhos

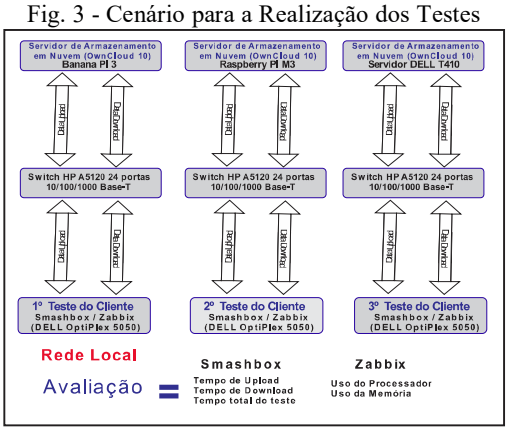
Artigo	Fog Computing	Plataforma Cloud	Dispositivo Embarcado	Benchmark	StaaS
[16] ZHU (2013)	✓				
[2] CRACIUNESCU (2015)	✓	✓	✓		
[15] VERBA (2016)	✓	✓	✓	✓	
[3] FAN (2016)	✓			✓	
[5] HAO (2017)	✓	✓		✓	
[8] LI (2017)	✓			✓	
[11] OSANAIYE (2017)	✓				
[12] POPENTIU-VLADICESCU (2017)	✓				
ESTE TRABALHO	✓	✓	✓	✓	✓

Fonte: Própria do Autor (2018)

4. Coleta de Dados

Para realizar a prototipação deve-se pressupor a existência de um modelo computacional idêntico ao ambiente real de produção. Na literatura, uma boa descrição para análise de desempenho em sistemas de nuvem para fornecer serviço de armazenamento StaaS (*Storage as a Service*) foi encontrada, como exemplo é possível citar o trabalho de MRÓWCZYŃSKI *et al.* (2017) [10]. A figura 3 ilustra a arquitetura do cenário para a realização dos testes de avaliação.

Fig. 3 - Cenário para a Realização dos Testes



Fonte: Própria do Autor (2018)

Tabela 2 - Abreviaturas dos Testes

Nome	Abreviação	Quantidade de arquivos	Tamanho arquivo	Volume total
Test0	0/1/1	1	1B	1B
Test1	0/1/100000000	1	100Mb	100Mb
Test2	0/10/10000000	10	10Mb	100Mb
Test3	0/1000/10000	1000	10Kb	10Mb
Test4	0/1/500000000	1	500Mb	500Mb

Fonte: Própria do Autor (2018)

Foram realizados cinco tipos de diferentes testes para avaliar o desempenho dos equipamentos analisados, para o fornecimento do serviço StaaS de forma sequencialmente. A tabela 2 informa abreviaturas dos testes (onde-se, 1º número equivale a quantidade de diretório, 2º quantidade de arquivo e o 3º volume do arquivo) e sua correspondente distribuição de arquivos.

4.1 Benchmark Smashbox

O Smashbox é uma estrutura de *benchmarking* e monitoramento para sincronização de arquivos e serviços de compartilhamento, permitindo aos provedores de serviços monitorar o status operacional de seus serviços, entendendo o comportamento do serviço sob diferentes tipos de carga e com diferentes locais de rede para a sincronização de clientes [10]. O *container* do sistema Smashbox está disponibilizado na condição pública no Docker Hub, no endereço https://hub.docker. com/r/ jsmac/smashbox/.

5. Avaliação e Resultados

Essa etapa consiste em: realizar a montagem do cenário de teste com os dispositivos de sistemas embarcados e o servidor de forma individual; efetuar a abordagem dos *softwares* necessários nos dispositivos para a elaboração do experimento; proceder com o processo de coleta dos dados do tempo de *upload*, tempo de *download* e tempo total de cada teste realizado com o *benchmark* Smashbox, as métricas de utilização da CPU e memória, equivalem ao máximo obtidos nas 30 repetições dos testes. A figura 4 ilustra o cenário real em que os testes foram realizados.

Fig. 4 - Cenário Real de Teste

Fonte: Própria do Autor (2018)

Tabela 3 - Diferentes Implementações

Equipamentos utilizados na avaliação				
EQUIP.	**S.O**	**VIRTUAL**	**RAM**	**REDE**
Banana PI M3	Ubuntu Server 16.04	Não suporta	2 Gb	1000 Mb/s
Raspberry PI 3	Raspbian Stretch Lite 9	S/ virtualização	1 Gb	100 Mb/s
Raspberry PI 3	Raspbian Stretch Lite 9	Docker	1 Gb	100 Mb/s
Raspberry PI 3	Ubuntu MATE 16.04	Docker	1 Gb	100 Mb/s
Servidor Dell T410	Ubuntu Server 16.04	VMware ESXi + Docker	16 Gb	1000 Mb/s
Desktop Dell OptiPlex 5050	Windows 10	Docker	16 Gb	1000 Mb/s

Fonte: Própria do Autor (2018)

Foram implementados 5 (cinco) cenários com diferentes sistemas operacionais, com e sem a implementação da virtualização Docker *Container*, isso possibilitou analisar o melhor conjunto implementado e o impacto da virtualização em dispositivos de sistemas embarcados. A tabela 3 mostra as diferentes implementações utilizadas com diferentes sistemas operacionais.

Totalizaram-se 750 coletas de dados, 30 repetições para os 5 diferentes testes, utilizando as 5 diferentes implementações entre os equipamentos testados e diferentes sistemas operacionais. Só foram aceitos para análise os testes concluídos sem erros. O tempo encontra-se em segundos. A tabela 4 mostra os resultados obtidos por todas as implementações para o Test0 na transferência de 1 arquivo do tamanho de 1 byte.

5.1 Test0 0/1/1

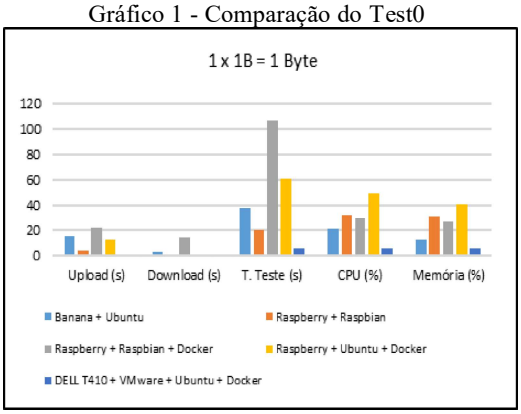
Nota-se que nesse teste todas as implementações obtiveram resultados melhores no tempo de *download* em relação ao tempo de *upload*. Também se observa que as implementações nos dispositivos de sistemas embarcados obtiveram um valor muito elevado na métrica de desvio padrão, o valor. O gráfico 1 mostra melhor visualmente a comparação entre todas as implementações.

Tabela 4 – Resultado do Test0

Tempo segundos	SMASHBOX				ZABBIX	
	MÉDIA	DESVIO	MÍN	MAX	CPU %	MEM %
BANANA PI M3 + UBUNTU SERVER						
T. UPL	15,13	5,54	3,00	26,00		
T. DWL	3,33	2,01	2,00	9,00	21,50	12,50
TOTAL	38,23	8,65	21,00	57,00		
RASPBERRY PI 3 + RASPBIAN STRETCH LITE						
T. UPL	4,20	5,26	1,00	17,00		
T. DWL	1,73	2,30	0,82	13,00	32,42	31,39
TOTAL	20,40	22,23	6,00	62,00		
RASPBERRY PI 3 + RASPBIAN LITE + DOCKER						
T. UPL	22,60	3,76	14,00	31,00		
T. DWL	14,53	2,61	7,00	20,00	29,36	26,55
TOTAL	106,60	10,41	73,00	122,00		
RASPBERRY PI 3 + UBUNTU MATE + DOCKER						
T. UPL	12,83	1,64	9,00	17,00		
T. DWL	7,03	1,27	4,00	11,00	49,93	41,09
TOTAL	60,87	2,15	56,00	64,00		
DELL T410 + VMWARE + UBUNTU + DOCKER						
T. UPL	1,00	0,00	1,00	1,00		
T. DWL	0,99	0,02	0,90	1,00	6,40	6,00
TOTAL	6,00	0,26	5,00	7,00		

Fonte: Própria do Autor (2018)

Gráfico 1 - Comparação do Test0



Fonte: Própria do Autor (2018)

A implementação do servidor DELL T410 é considerada a ideal, afinal é um equipamento de alto desempenho, porém com um alto custo aquisitivo, percebe-se que os seus resultados foram melhores em quase todos os aspectos analisados em relação às outras implementações.

5.2 Test1 0/1/100000000

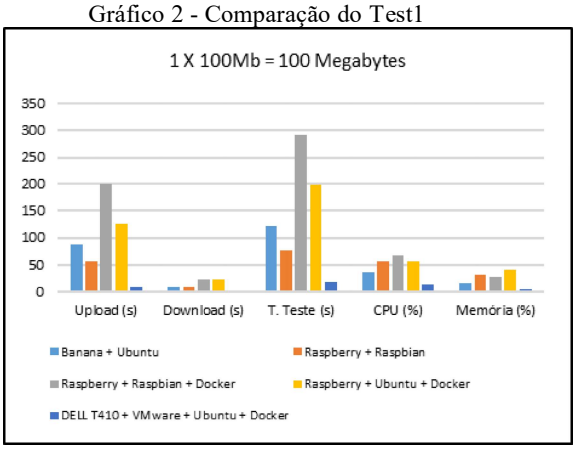
O Test1 consiste em avaliar a transferência do volume de um arquivo com o tamanho de 100 megabytes. A tabela 5 mostra os resultados obtidos por todas as implementações para o Test1 na transferência do arquivo.

Tabela 5 – Resultado do Test1

Tempo segundos	SMASHBOX				ZABBIX	
	MÉDIA	DESVIO	MÍN	MAX	CPU %	MEM %
BANANA PI M3 + UBUNTU SERVER						
T. UPL	88,50	14,45	67,00	125,00		
T. DWL	10,60	4,58	6,00	27,00	36,31	15,50
TOTAL	121,63	19,24	95,00	159,00		
RASPBERRY PI 3 + RASPBIAN STRETCH LITE						
T. UPL	56,93	26,51	30,00	144,00		
T. DWL	9,70	1,78	9,00	18,00	56,21	33,06
TOTAL	77,67	34,84	44,00	191,00		
RASPBERRY PI 3 + RASPBIAN LITE + DOCKER						
T. UPL	199,20	16,61	148,00	223,00		
T. DWL	23,93	5,56	14,00	42,00	67,60	28,76
TOTAL	290,90	25,80	174,00	317,00		
RASPBERRY PI 3 + UBUNTU MATE + DOCKER						
T. UPL	125,13	7,29	117,00	153,00		
T. DWL	23,57	4,28	17,00	32,00	55,96	40,43
TOTAL	197,13	9,67	184,00	237,00		
DELL T410 + VMWARE + UBUNTU + DOCKER						
T. UPL	9,23	0,43	9,00	10,00		
T. DWL	4,00	0	4,00	4,00	14,03	6,00
TOTAL	18,87	0,35	18,00	19,00		

Fonte: Própria do Autor (2018)

Gráfico 2 - Comparação do Test1



Fonte: Própria do Autor (2018)

A implementação Banana + Ubuntu obteve uma métrica melhor no quesito tempo médio de *upload* comparado a implementação do Raspberry com ubuntu e utilizando virtualização Docker, fato que não tinha ocorrido no teste Test0. Porém, seus resultados ficaram abaixo comparado com a implementação do Raspberry com raspbian sem virtualização. O gráfico 2 mostra melhor visualmente a comparação.

5.3 Test2 0/10/10000000

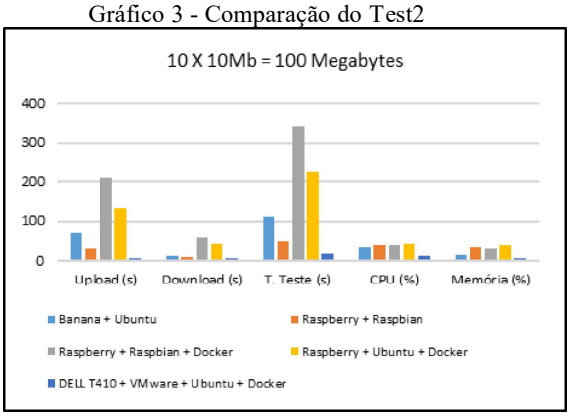
O test2 consiste em avaliar a transferência do volume de dez arquivos com o tamanho de 10 megabytes. A tabela 6 mostra os resultados obtidos por todas as implementações para o Test2 na transferência dos arquivos.

Tabela 6 – Resultado do Test2

Tempo segundos	SMASHBOX				ZABBIX	
	MÉDIA	DESVIO	MÍN	MAX	CPU %	MEM %
BANANA PI M3 + UBUNTU SERVER						
T. UPL	70,83	13,28	49,00	111,00		
T. DWL	11,77	7,54	5,00	32,00	34,91	15,50
TOTAL	111,17	18,20	85,00	147,00		
RASPBERRY PI 3 + RASPBIAN STRETCH LITE						
T. UPL	30,63	29,74	16,00	158,00		
T. DWL	10,20	3,92	9,00	30,00	40,88	33,85
TOTAL	49,83	34,27	33,00	199,00		
RASPBERRY PI 3 + RASPBIAN LITE + DOCKER						
T. UPL	209,63	14,72	184,00	236,00		
T. DWL	59,37	8,273	50,00	95,00	41,13	30,37
TOTAL	341,77	19,49	307,00	414,00		
RASPBERRY PI 3 + UBUNTU MATE + DOCKER						
T. UPL	133,37	21,92	117,00	242,00		
T. DWL	42,40	7,96	34,00	80,00	41,96	39,50
TOTAL	225,70	21,93	212,00	330,00		
DELL T410 + VMWARE + UBUNTU + DOCKER						
T. UPL	7,40	0,89	6,00	9,00		
T. DWL	4,80	0,61	4,00	6,00	13,20	6,00
TOTAL	18,73	1,08	17,00	21,00		

Fonte: Própria do Autor (2018)

Gráfico 3 - Comparação do Test2



Fonte: Própria do Autor (2018)

Na implementação Raspberry + Raspbian + Docker, nota-se que continuou obtendo os piores resultados no teste, com a média 209,63 segundos de *upload* e 59,37 segundos de *download*, e uma média de tempo de conclusão dos testes muito elevada 341,77 segundos. Percebe-se também que a implementação chegou a utilizar um nível razoável de processamento no teste 41,13% e com o uso de memória mediano 30,37%. O gráfico 3 mostra melhor visualmente a comparação.

5.4 Test3 0/1000/10000

O test3 consiste em avaliar a transferência do volume de mil arquivos com o tamanho de 10 kilobytes, acredita-se que esse teste seja a simulação mais próxima da realidade do funcionamento de uma *Fog Computing*, pequenos arquivos, porém em grande quantidade. A tabela 7 mostra os resultados obtidos por todas as implementações.

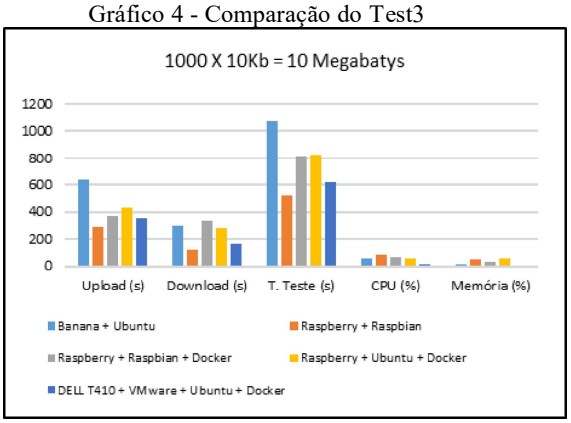
A implementação Raspberry + Raspbian surpreendeu, superando até mesmo a implementação do servidor DELL T410. Todavia nota-se que o desvio padrão dos seus resultados apresentaram uma alta discrepância, isso o torna um serviço pouco preciso. O gráfico 4 mostra melhor visualmente a comparação.

Tabela 7 – Resultado do Test3

Tempo segundos	SMASHBOX				ZABBIX	
	MÉDIA	DESVIO	MÍN	MAX	CPU %	MEM %
BANANA PI M3 + UBUNTU SERVER						
T. UPL	640,67	459,94	418,00	2496,00		
T. DWL	301,67	121,19	232,00	687,00	59,67	18,00
TOTAL	1079,03	540,16	804,00	2889,00		
RASPBERRY PI 3 + RASPBIAN STRETCH LITE						
T. UPL	289,40	174,19	148,00	746,00		
T. DWL	120,77	98,14	87,00	546,00	87,21	46,56
TOTAL	521,50	212,90	346,00	1175,00		
RASPBERRY PI 3 + RASPBIAN LITE + DOCKER						
T. UPL	369,03	46,41	346,00	610,00		
T. DWL	334,43	25,82	327,00	469,00	67,65	31,14
TOTAL	812,60	56,57	779,00	1080,00		
RASPBERRY PI 3 + UBUNTU MATE + DOCKER						
T. UPL	430,57	81,83	335,00	685,00		
T. DWL	278,13	210,58	106,00	1056,00	61,65	60,82
TOTAL	821,17	292,95	553,00	1858,00		
DELL T410 + VMWARE + UBUNTU + DOCKER						
T. UPL	350,63	6,14	340,00	364,00		
T. DWL	163,93	0,25	163,00	164,00	12,00	7,08
TOTAL	620,63	6,36	610,00	635,00		

Fonte: Própria do Autor (2018)

Gráfico 4 - Comparação do Test3



Fonte: Própria do Autor (2018)

5.5 Test4 0/1/500000000

O Test4 consiste em avaliar a transferência do volume de um arquivo com o tamanho de 500 megabytes. A tabela 8 mostra os resultados obtidos por todas as implementações.

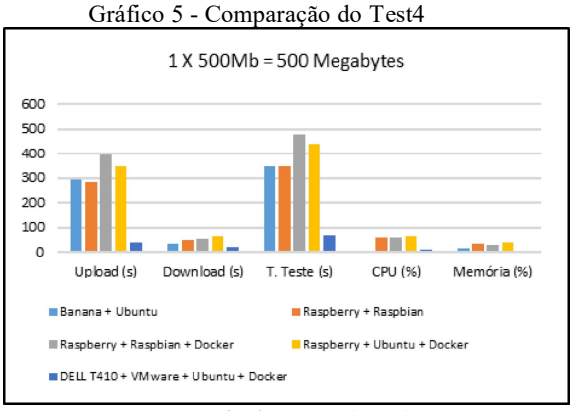
Na implementação Raspberry + Ubuntu + Docker obteve bons resultados quando comparado com a implementação de concorrência direta, Raspberry + Raspbian + Docker. Todavia nota-se que foi a implementação que obteve o mais alto nível do uso da memória do *hardware* 65,00%. O gráfico 5 mostra melhor visualmente a comparação.

Tabela 8 – Resultado do Test4

Tempo segundos	SMASHBOX				ZABBIX	
	MÉDIA	DESVIO	MÍN	MAX	CPU %	MEM %
BANANA PI M3 + UBUNTU SERVER						
T. UPL	294,00	24,62	237,00	340,00		
T. DWL	36,33	12,69	21,00	81,00	40,92	17,00
TOTAL	347,47	31,03	271,00	394,00		
RASPBERRY PI 3 + RASPBIAN STRETCH LITE						
T. UPL	281,83	84,91	174,00	449,00		
T. DWL	49,40	5,93	44,00	69,00	57,05	33,96
TOTAL	345,43	94,24	226,00	544,00		
RASPBERRY PI 3 + RASPBIAN LITE + DOCKER						
T. UPL	394,37	146,64	191,00	886,00		
T. DWL	55,30	8,26	47,00	81,00	59,41	30,30
TOTAL	476,80	158,12	254,00	1018,00		
RASPBERRY PI 3 + UBUNTU MATE + DOCKER						
T. UPL	346,30	168,15	190,00	850,00		
T. DWL	63,97	25,32	48,00	161,00	65,00	38,95
TOTAL	437,03	188,62	258,00	961,00		
DELL T410 + VMWARE + UBUNTU + DOCKER						
T. UPL	40,30	3,48	37,00	51,00		
T. DWL	19,23	3,08	16,00	27,00	12,00	7,08
TOTAL	67,47	5,96	61,00	80,00		

Fonte: Própria do Autor (2018)

Gráfico 5 - Comparação do Test4



Fonte: Própria do Autor (2018)

Diferentes aplicações de computação em *Fog* foram sugeridas na literatura. Segundo OSANAIYE *et al.* (2017) [11], as categorias das aplicações de computação em *Fog* são divididas em 3: (i) Aplicações em tempo real; (ii) Aplicações quase em tempo real; (iii) Aplicações introduzidas em redes.

Por isso, definir o nível aceitável do fornecimento do serviço de armazenamento StaaS, em uma emergente tecnologia em que se trata o ambiente *Fog Computing* é uma tarefa complexa e subjetiva, dependente da classificação da aplicação.

6. Conclusão e Trabalhos Futuros

Este trabalho apresenta o conceito da *Fog Computing*, sua contextualização teórica, os trabalhos correlatos, realiza análise de uma *Fog Computing*, para fornecer StaaS (*Storage as a Service*), a dispositivos IoT utilizando plataformas de sistemas embarcados e compara seus resultados com os obtidos por um servidor de alto desempenho. Foram realizados cinco (5) implementações de diferentes combinações de *softwares* e *hardwares*, e analisa seus resultados com a finalidade de encontrar a melhor opção para disponibilizar o serviço de armazenamento StaaS em um ambiente *Fog Computing*.

Os resultados satisfizeram as expectativas, na avaliação do teste Test3 0/1000/10000 (transferência de 1000 arquivos de 10 *Kilobytes*) a implementação Raspberry + Raspbian surpreendeu, obtendo ótimos resultados, superando até mesmo a implementação do servidor DELL T410. Nota-se que este tipo de implementação pode satisfazer as aplicações classificadas em aplicações quase em tempo real e introduzida na rede, não sendo adequada para as aplicações classificadas como aplicações em tempo real, devido as implementações nos dispositivos de sistemas embarcados obterem valores muito elevados nas métricas de desvio padrão, tornando o serviço pouco preciso.

Portanto percebe-se que a implementação desse serviço em dispositivos de sistemas embarcados pode ser uma boa alternativa para reduzir um desses problemas, no caso, o armazenamento de dados, que atinge hoje os dispositivos IoT, servindo como *Fog Computing* e sendo implantado num dispositivo de plataforma embarcada de baixo custo, ao invés de usar potentes e caros servidores para exercer essa função.

6.1 Trabalhos Futuros

Com a finalidade em dar continuidade a essa pesquisa, acredita-se que esse trabalho abriu vários cenários para futuros trabalhos, sendo:

- Analisar o serviço StaaS, com diferentes dispositivos embarcados e diferentes plataformas de serviço de armazenamento em nuvem, não utilizados neste trabalho;
- Realizar a mesma avaliação utilizando *cluster* com dispositivos de sistemas embarcados, usando a virtualização Dorcker.

7. Referências

[1] AL-DOGHMAN, Firas et al. A review on Fog Computing technology. In: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on. IEEE, 2016. p. 001525-001530.

[2] CRACIUNESCU, Razvan et al. Implementation of Fog computing for reliable E-health applications. In: Signals, Systems and Computers, 2015 49th Asilomar Conference on. IEEE, 2015. p. 459-463.

[3] FAN, Chih-Tien et al. Web Resource Cacheable Edge Device in Fog Computing. In: Parallel and Distributed Computing (ISPDC), 2016 15th International Symposium on. IEEE, 2016. p. 432-439.

[4] HAJIBABA, Majid; GORGIN, Saeid. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. CIT. Journal of Computing and Information Technology, v. 22, n. 2, p. 69-84, 2014.

[5] HAO, Zijiang et al. Challenges and Software Architecture for Fog Computing. IEEE Internet Computing, v. 21, n. 2, p. 44-53, 2017.

[6] JAIN, Akshay; SINGHAL, Priyank. Fog computing: Driving force behind the emergence of edge computing. In: System Modeling & Advancement in Research Trends (SMART), International Conference. IEEE, 2016. p. 294-297.

[7] MACHADO, José dos Santos; MORENO, Edward David; RIBEIRO, Admilson de Ribamar Lima. A Review of Computing Fog and its Research Challenges. Journal on Advances in Theoretical and Applied Informatics, [S.l.], v. 3, n. 2, p. 32-39, dec. 2017. ISSN 2447-5033.

[8] LI, Songze; MADDAH-ALI, Mohammad Ali; AVESTIMEHR, A. Salman. Coding for Distributed Fog Computing. IEEE Communications Magazine, v. 55, n. 4, p. 34-40, 2017.

[9] MARTINI, Ben; CHOO, Kim-Kwang Raymond. Cloud storage forensics: ownCloud as a case study. Digital Investigation, v. 10, n. 4, p. 287-299, 2013.

[10] MRÓWCZYŃSKI, Piotr et al. Benchmarking and monitoring framework for interconnected file synchronization and sharing services. Future Generation Computer Systems, 2017.

[11] OSANAIYE, Opeyemi et al. From cloud to fog computing: A review and a conceptual live VM migration framework. IEEE Access, 2017.

[12] POPENTIU-VLADICESCU, Florin; ALBEANU, Grigore. Software reliability in the fog computing. In: Innovations in Electrical Engineering and Computational Technologies (ICIEECT), 2017 International Conference on. IEEE, 2017. p. 1-4.

[13] MCMILLIN, Bruce; ZHANG, Tao. Fog Computing for Smart Living. Computer, v. 50, n. 2, p. 5-5, 2017.

[14] STOJMENOVIC, Ivan et al. An overview of Fog computing and its security issues. Concurrency and Computation: Practice and Experience, 2015.

[15] VERBA, Nandor et al. Platform as a service gateway for the Fog of Things. Advanced Engineering Informatics, 2016.

[16] ZHU, Jiang et al. Improving web sites performance using edge servers in fog computing architecture. In: Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on. IEEE, 2013. p. 320-323.

[17] ALRAWAIS, Arwa et al. Fog Computing for the Internet of Things: Security and Privacy Issues. IEEE Internet Computing, v. 21, n. 2, p. 34-42, 2017.

[18] BABU, Shaik Masthan; LAKSHMI, A. Jaya; RAO, B. Thirumala. A study on cloud based internet of things: Cloudiot. In: Communication Technologies (GCCT), 2015 Global Conference on. IEEE, 2015. p. 60-65.

[19] BOTTA, Alessio et al. Integration of cloud computing and internet of things: a survey. Future Generation Computer Systems, v. 56, p. 684-700, 2016.

[20] CHEN, Songqing; ZHANG, Tao; SHI, Weisong. Fog Computing. IEEE Internet Computing, v. 21, n. 2, p. 4-6, 2017.

[21] CHIANG, Mung et al. Clarifying Fog Computing and Networking: 10 Questions and Answers. IEEE Communications Magazine, v. 55, n. 4, p. 18-20, 2017.

[22] DÍAZ, Manuel; MARTÍN, Cristian; RUBIO, Bartolomé. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. Journal of Network and Computer Applications, v. 67, p. 99-117, 2016.

[23] KUMAR, Praveen; ZAIDI, Nabeel; CHOUDHURY, Tanupriya. Fog computing: Common security issues and proposed countermeasures. In: System Modeling & Advancement in Research Trends (SMART), International Conference. IEEE, 2016. p. 311-315.

[24] LINTHICUM, David S. Connecting Fog and Cloud Computing. IEEE Cloud Computing, v. 4, n. 2, p. 18-20, 2017.

[25] NWOBODO, Ikechukwu (2015). A Comparison of Cloud Computing Platforms. International Symposium on Circuits and Systems (ISCAS). Lisbon, Portugal, 24-27 May 2015. Atlantis Press.

[26] PRINCY, S. Emima; NIGEL, K. Gerard Joe. Implementation of cloud server for real time data storage using Raspberry Pi. In: Green Engineering and Technologies (IC-GET), 2015 Online International Conference on. IEEE, 2015. p. 1-4.

[27] LONGO, Francesco et al. Stack4things: An openstack-based framework for iot. In: Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on. IEEE, 2015. p. 204-211.

[28] ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, v. 1, n. 1, p. 7-18, 2010.

Internet of Things: A Survey on Communication Protocol Security

Walter E. Santo
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
walterdoespiritosanto@gmail.com

Ricardo J. P. de B. Salgueiro
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
salgueiro@ufs.br

Reneilson Santos
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
reneilson1@gmail.com

Danilo Souza
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
danilosilva.se@gmail.com

Admilson Ribeiro
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
admilson@ufs.br

Edward Moreno
Universidade Federal de Sergipe
Sao Cristovao, Sergipe
edwdavid@gmail.com

ABSTRACT

This paper presents a survey on the main security problems that affect the communication protocols in the context of Internet of Things, in order to identify possible threats and vulnerabilities. The protocols RFID, NFC, 6LoWPAN, 6TiSCH, DTSL, CoAP and MQTT, for a better organization, were explored and categorized in layers according to the TCP / IP reference model. At the end, a summary is presented in tabular form with the security modes used for each protocol is used.

CCS CONCEPTS

• **Networks** → Network protocols; Security protocols; Cyber-physical networks; • **General and reference** → Surveys and overviews;

KEYWORDS

Internet of Things, Vulnerabilities, Security, IoT Protocols, Survey

ACM Reference Format:
Walter E. Santo, Ricardo J. P. de B. Salgueiro, Reneilson Santos, Danilo Souza, Admilson Ribeiro, and Edward Moreno. 2018. Internet of Things: A Survey on Communication Protocol Security. In *Proceedings of Euro-American Conference on Telematics and Information Systems (EATIS'18).* ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.475/123_4

1 INTRODUÇÃO

A Internet das Coisas, do inglês: Internet of Things (IoT), denominação sugerida por Kevin Aston, é uma revolução tecnológica em computação e comunicações que retrata uma variedade de dispositivos inteligentes amplamente interconectados [17] e tem uma entidade digital [10]. Os dispositivos são capazes de interpretar e reagir ao ambiente graças à combinação da Internet com tecnologias emergentes como a Identificação por Radiofrequência (RFID) [7], localização em tempo real e sensores embarcados. Até 2020, estima-se que 13.5 bilhões de dispositivos estarão conectados [4].

EATIS'18, November 2018, Fortaleza - Ceará, Brasil

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

O aumento do número de dispositivos IoT deve ser acompanhado por uma infraestrutura capaz de suportar a enorme quantidade de tráfego, armazenamento e processamento dos dados gerados, de maneira eficiente e segura. Conhecer quais são os protocolos e suas principais características é extremamente importante ao se projetar a arquitetura do ambiente IoT, de modo a garantir sua segurança. Falhas de transmissão, negação de serviço, interceptação dos dados, ataques de autenticação, *spoofing*, entre outros, podem vir a acontecer caso a escolha do protocolo não esteja condizente com as especificações e limitações dos dispositivos e das diversas interfaces com as quais eles se comunicam. Uma comunicação segura envolve confidencialidade, integridade, autenticação e não-repudio, que podem ser endereçadas pelos protocolos ou por mecanismos externos [12]

Neste trabalho, é fornecido um *survey* com ênfase nas principais características e garantias de segurança com baixo consumo computacional nos protocolos de comunicação da Internet das Coisas, alertando para possíveis vetores de ataque. Este está estruturado em cinco seções: a Seção 2 exibe trabalhos relacionados; Seção 3 apresenta a segurança nos principais protocolos IoT; a Seção 4 é dedicada à síntese de segurança dos protocolos; por fim, na Seção 5 tem-se as conclusões, com sugestões de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Estudos relacionados à área de segurança tentam fornecer variados levantamentos em diferente aspectos relacionados a segurança e protocolos de comunicação em IoT.

Lin et. al [14], discutem como a IoT pode se beneficiar com o uso da *Fog Computing* e apresentam uma abrangente pesquisa de esforços recentes sobre arquiteturas, tecnologias facilitadoras e questões de segurança e privacidade. Yang et. al em [26] exploram a segurança em IoT do ponto de vista da limitação dos dispositivos, autenticação e controle de acesso. Além disso, os autores apresentam a classificação de diferentes ataques e discutem perspectivas da segurança em diferentes camadas da arquitetura IoT. Ammar et al. [9], apresentam uma análise comparativa dos principais frameworks de IoT para sustentar o argumento de que o sucesso no desenvolvimento de aplicações IoT depende principalmente das questões relacionadas à segurança e privacidade. Os resultados da comparação apontam que as arquiteturas de segurança utilizam padrões de proteção semelhantes embora adotem metodologias

diferentes. O trabalho apresentado por [15], discute aspectos do uso de *middlewares* para IoT. Não obstante, os autores apresentam uma análise dos desafios e das tecnologias capacitadoras no desenvolvimento de um *middleware* IoT considerando aspectos como heterogeneidade de dispositivos, adaptabilidade e segurança.

Diferente dos recentes esforços de pesquisa à respeito segurança em IoT, o presente trabalho dá ênfase em aspectos de segurança dos principais protocolos IoT e suas camadas, bem como, discute a implementação da segurança e diferentes tipos de ataques.

3 SEGURANÇA NOS PROTOCOLOS IOT

Os principais problemas de segurança relacionados à IoT envolvem questões relativas à privacidade, dessa forma, devem ser orquestradas soluções para estes problemas a partir da estipulação de protocolos [23], que devem prover mecanismos de segurança, ao mesmo tempo que fornecem a agilidade e escalabilidade necessária para o fluxo de dados [16].

Quando se trata de IoT, diversos fatores influenciam a escolha dos protocolos a serem utilizados. Tempo de vida da bateria, *Ligthweight Computation* [26], necessidades da troca de dados, alcance mínimo e máximo, mobilidade dos nós na rede, taxas de perda e de erro, comunicação com a nuvem, entre outros. Além de utilizar os protocolos já conhecidos da Internet convencional como TCP/IP, HTTP/REST, WiFi e Ethernet, novos protocolos ganham importância, principalmente nas camadas físicas e de enlace, nas quais os dispositivos, conectados a sensores, formam as RSSF (Redes de Sensores sem Fio) e passam a possuir restrições energéticas e de processamento.

Para o presente estudo, os protocolos foram categorizados de acordo com as camadas as quais pertencem. A Tabela 1 ilustra o posicionamento dos protocolos que são abordados de acordo com suas camadas principais no modelo de referência TCP/IP.

Table 1: Categorização dos protocolos em camadas segundo o modelo TCP/IP

Modelo de referência TCP/IP	Protocolos
4- Aplicação	CoAP, MQTT
3- Transporte	DTLS
2- Rede	6loWPAN, 6TiSCH
1- Física/Enlace	RFID, NFC

3.1 Camada Física e de Enlace: RFID e NFC

Considerado como base fundamental para a definição do que é a Internet das Coisas, a identificação por rádio frequência (RFID) [8], se apresenta como uma solução para o endereçamento único de dispositivos. Funciona pela emissão de ondas eletromagnéticas por leitores que, por sua vez, ativam as etiquetas, que contêm informações elétricas armazenadas e as transmitem por uma antena. Tais etiquetas podem ser passivas (ativadas no momento em que recebem o estímulo da onda eletromagnética), ou ativas (ligadas a uma fonte de energia, possuindo, por conseguinte, um alcance maior). Os principais aspectos de segurança dos protocolos de baixo custo utilizados para o RFID, segundo [27] são: confidencialidade, integridade, disponibilidade, autenticidade e privacidade.

O *Near Field Communication* (NFC) [11], é um conjunto de protocolos para a comunicação de dispositivos fisicamente próximos através de campo eletromagnético (incluindo o RFID). Entretanto, o NFC é definido apenas para objetos com aproximadamente 10cm de distância e não possui restrições quanto à direcionalidade da comunicação (uma etiqueta pode se comportar como um leitor e o leitor como etiqueta), possibilitando uma comunicação ponto a ponto [1]. A principal utilização do NFC tem sido para pagamentos sem cartão.

Definido pelo padrão ISO-14443, o protocolo NFC possui três fases principais: (i) Evitar Colisão de RF (Rádio Frequência) – o Leitor só ativa sua RF quando nenhuma outra RF tiver sido detectada; (ii) Detecção de Dispositivo – o Leitor sonda alvos próximos e recebe uma resposta em determinado intervalo de tempo (*time-slot*); e (iii) Protocolo de Transporte – após ter encontrado um alvo, o Leitor inicia a transmissão utilizando do protocolo de transporte, o qual especifica parâmetros como o *timeout* esperado [6].

A Tabela 2 traz uma lista dos principais e mais comuns ataques aos protocolos RFID e NFC.

Table 2: Principais e mais comuns ataques aos protocolos RFID e NFC

Tipo de ataque	Descrição
Eavesdropping	Ocorre quando um espião consegue ter acesso à informação transmitida entre uma etiqueta e um leitor.
Ataques *Man-In-The-Middle*	Um atacante entre um servidor e uma etiqueta recebe os dados da comunicação sendo realizada.
Ataques de *Replay*	São proporcionados por atacantes que têm acesso a um dado transmitido e o repassa com *spoofing* da identificação da etiqueta e também para ataques *Man-in-the-middle*.
Ataques de Desincronização	Um tipo de ataque de negação de serviço em que a informação relativa a uma etiqueta armazenada em um servidor é confundida com a informação que está armazenada na etiqueta, inviabilizando a comunicação.
Ataques de Personificação	Um atacante faz uso da identificação da etiqueta para se autenticar em um servidor.
Ataques de Negação de Serviço (DoS)	É adicionado ruído de modo a interromper a operação normal do RFID.
Jamming físico	Funciona como um ataque de DoS, ao se transmitir sinais de rádio que impõem ruído ao sinal transmitido.

3.2 Camadas de Rede: 6LoWPAN e 6TiSCH

Como não é possível se estabelecer uma integração direta entre o IPv6 e o IEEE 802.15.4 [1], o grupo IPv6 sobre redes sem-fio de baixa potência em PANs, do inglês *IPv6 over Low-power Wireless*

[1]O protocolo IEEE 802.15.4 define quais são as condições necessárias para a comunicação entre dispositivos com baixo consumo energético e de pouco alcance, as *Low-power Wireless Personal Area Network* (LoWPAN)

Personal Area Networks (6LoWPAN)[13], busca mecanismos para o desenvolvimento de uma pilha de protocolos que forneça essa integração. Técnicas como compressão de cabeçalho, fragmentação e reestruturação de pacotes, descoberta de vizinhos e autoconfiguração são utilizadas na adequação do protocolo IPv6 para as redes sem fio de baixa potência em WPANs.

O 6LoWPAN, então, define uma nova camada de adaptação entre a camada de rede e de enlace (*6LoWPAN Adaptation Layer*), na qual é realizada: a fragmentação e reconstrução dos pacotes enviados, que não podem ser fragmentados pelo IPv6; a compressão do cabeçalho e o roteamento para a camada de enlace. Granjal et al. em [12] ressalta que não são implementados mecanismos de segurança específicos para o 6LoWPAN, visto que este conta com a segurança a nível de enlace provida pelo protocolo IEEE 802.15.4. O fato de não estar autenticado, permite que atacantes explorem vulnerabilidades no processo de fragmentação, no qual deve ser mantido um *buffer* para a remontagem dos pacotes. Também, como os dados não são criptografados, em [18] afirma-se que o 6LowPAN está vulnerável a ataques de *eavesdropping*, *man-in-the-middle* e *spoofing*.

Um estudo realizado por Vohra e Srivastava em [25] elicita trabalhos de pesquisa de vulnerabilidades no 6LoWPAN e técnicas para a segurança no mesmo, porém, não se mostrou, até o momento, adequada para redes de baixo custo energético e de processamento. Ainda para a integração, o grupo 6TiSCH busca integrar o IPv6 com a versão IEEE 802.15.4e. Nele, é realizada a divisão de tempo TDMA (Acesso Múltiplo por Divisão de Tempo), em que uma faixa de banda é definida para a comunicação entre nós vizinhos [3].

3.3 Camada de Transporte: DTLS

O TLS é o protocolo utilizado para garantir segurança na Internet, desde aplicações bancárias até trocas de mensagens instantâneas, porém tem um alto custo computacional para ser implementado e não foi desenvolvido para aplicações de tempo crítico [11]. O DTLS (*Datagrama Transport Layer Security*) se apresenta como um protocolo para trazer a segurança na comunicação de datagramas pela rede através do UDP, que é menos confiável para a entrega de pacotes, porém permite um maior fluxo de dados, para superar o problema de tempo crítico. O DTLS é uma alternativa viável ao TLS no que se refere à transferência de dados pela Internet de Redes de Sensores Sem Fio (RSSF) conectadas. Apesar de ter sido desenvolvido para superar questões relativas ao tempo-crítico na Internet convencional, como no caso de games, por exemplo, o protocolo DTLS ganha especial importância para IoT.

O TLS funciona sobre quatro outros protocolos: (1) *Record Protocol*; (2) *Handshake*[2] *Protocol*; (3) *Alert Protocol* e (4) *Change Cipher Spec.* [12].

Um estudo exposto no RFC 7457[3] traz as principais vulnerabilidades conhecidas para o TLS e DTLS [20] - *stripping*; injeção de comandos STARTTLS; ataque BEAST; ataque *padding oracle*; ataques no RC4; ataques de compressão: CRIME, TIME, e *Breach*; ataques de certificado e RSA; parâmetros *diffie-hellman*; roubo de chaves privadas do RSA; renegociação; ataque de *handshake* triplo; confusão do hospedeiro Virtual; negação de serviço (DoS); problema de implementação e problema de usabilidade.

[2] *Handshake*;
[3] RFC (*Request for Comments*) 7457 <https://tools.ietf.org/html/rfc7457>

Versões antigas do TLS devem ser evitadas, pois são consideradas inseguras. Atualmente, a versão 1.2, tanto do TSL como DTLS, são preferíveis, e é importante não deixar que o protocolo caia de versão, processo que pode ser ativado por um ataque *Man-In-The-Middle*, colocando o sistema em posição instável, dada a insegurança das versões antigas [20].

3.4 Camada de Aplicação: CoAP e MQTT

Definido pela IETF (*Internet Engineering Task Force*), o CoAP (*The Constrained Application Protocol*) [21] representa o protocolo de camada de aplicação para redes e nós com restrições. É definido para aplicações M2M (*Machine to Machine*) e assemelha-se ao HTTP. Utiliza comandos GET, PUT, POST e DELETE, do modelo REST, e faz uso de conceitos da web como URIs [2]. A implementação do CoAP, porém, se comporta tanto como servidor como cliente em uma comunicação M2M.

O CoAP é dividido em duas camadas, uma que lida com requisições e respostas e outra para tratar as mensagens sendo transmitidas pelo UDP. Existem quatro possíveis tipos de mensagem no CoAP: (1) *Acknowledgement* (ACK), para sucesso; (2) *Reset*, para rejeitar uma mensagem confirmável ou remover um observador; (3) Confirmável, indica uma entrega confiável da mensagem e (4) Não-Confirmável, não espera uma confirmação do envio. Como está sobre o UDP, em que a entrega não é garantida, a transmissão pode exigir confirmação de entrega, por isso mensagens do tipo confirmável sempre retornam um ACK quando bem-sucedidas [19]. Assim como no HTTP, o CoAP possui sua série de códigos e mensagens de resposta [12].

Para a segurança, o CoAP utiliza do DTLS, logo, transfere para a camada de transporte a manipulação de mecanismos de segurança [12]. O protocolo provê quatro modos de segurança: (1) *NoSec*: nenhum mecanismo de segurança do DTLS é aplicado, (2) *PreSharedKey*: utilizado com dispositivos que já são pré-programados com as chaves simétricas necessárias, onde cada chave possui uma lista de nós que podem se comunicar, (3) *RawPublicKey*: o dispositivo possui um par de chaves assimétricas sem a utilização de certificado, que é validado por um mecanismo *out-of-band* e (4) *Certificate*: o protocolo faz o uso do DTLS com um certificado X.509, o dispositivo possui também usa uma lista de raízes confiáveis [12, 21].

A Tabela 3 traz as possíveis ameaças ao protocolo CoAP de acordo com o RFC 7252 [21].

O MQTT (*Message Queuing Telemetry Transport Protocol*), desenvolvido em 1999 originalmente pela IBM, se tornou um padrão aberto ISO (ISO/IEC 20922:2016) [5]. Trata-se de um protocolo, leve e simples de implementar, para o enfileiramento e transporte de mensagens, que se utiliza do modelo *publish/subscribe*. Seus componentes principais são: *brokers*, sessões, assinaturas (subscriptions) e assunto (*topic*) [22].

O modelo *publish/subscribe* envolve a definição de um comunicante e de diversos ouvintes, conectados em um *broker*, que organiza a troca de mensagens entre assinantes e publicantes. O assinante registra o interesse em determinado assunto e, assim que algum publicante disponibiliza conteúdo neste tópico, o *broker* direciona a mensagem para os assinantes registrados. A qualidade de serviço nesse processo é dividida em três categorias: (1) No máximo uma vez (*At most once/Fire and Forget*), que utiliza do melhor esforço

Table 3: Ameaças ao protocolo CoAP – RFC 7252

Tipo de ataque	Descrição
Parsing do Protocolo e Processamento de URIs	É possível explorar vulnerabilidades no processo de *parsing* (processo que analisa uma sequência de entrada), para, por exemplo, gerar um ataque de negação de serviço ao se inserir um texto que irá acarretar em parser muito extenso.
Proxying e *Caching*	O *proxy* é, por si só, um *man-in-the-middle*, quebrando toda segurança do IPSec e DTLS. Ameaças são amplificadas quando os *proxies* permitem que haja uma cache dos dados.
Risco de Amplificação	As respostas no CoAP são, geralmente, maiores do que as requisições, o que pode facilitar ataques por amplificação.
Ataques de IP *Spoofing*	Como não há *handshake* para o UDP, o nó final que possui acesso à rede pode realizar *spoofing* para enviar mensagens de ACK no lugar de CON, prevenindo que haja retransmissão; *spoofing* em todo o *payload*; *spoofing* de pedidos *multicast*; etc.
Ataques *Cross-Protocol*	Envolvem utilizar o CoAP para enviar ataques a outros protocolos, para se passar pelo *firewall*, por exemplo.
Nós com Restrições	Sejam restrições energéticas, de memória ou de processamento, dificultam que os dispositivos disponham de boa entropia. Assume-se, portanto, que os processos que necessitem de entropia, como o cálculo de chaves, o façam externamente.

para se enviar, caso não chegue em determinado envio pode chegar no próximo; (2) Ao menos uma vez (*At least once*), garante que a mensagem chega, mas pode ocorrer duplicatas e (3) Exatamente uma vez (*Exactly once*), que garante que a mensagem chegará e não irão ocorrer duplicatas [22].

Assim como no CoAP, a segurança é endereçada por fora, pelo TLS, que é muito pesado para os dispositivos IoT. Em [24] é proposto um modelo de aplicação seguro para o MQTT, denominado SMQTT, por meio de criptografia baseada em atributo leve (*Lightweight* ABE), que provê criptografia por *broadcast*, sobre curvas elípticas. Esse modelo, segundo os autores, se mostrou resistente a ataques de *plaintext* conhecido, *ciphertext* conhecido e de *man-in-the-middle*.

4 SÍNTESE DE SEGURANÇA DOS PROTOCOLOS

O presente trabalho teve seu foco principal na abordagem da segurança dentro dos principais protocolos de comunicação IoT, identificando as respectivas vulnerabilidades e ameaças. Nessa direção, como resultado do estudo realizado, criou-se uma compilação dos possíveis tipos de ataques bem como os modos de segurança empregados em cada protocolo, conforme pode ser visualizado na Tabela 4. Um agrupamento geral, reunindo as ameaças mais recorrentes, comuns aos principais protocolos, classificaram estes de acordo com as camadas a que pertenciam. No decorrer da pesquisa, observou-se a ausência de dados compilados dessa natureza, de fácil acesso e manipulação, voltados para segurança em IoT. Por conseguinte, o resultado desse estudo tem como principal diferencial fornecer um material consistente para pesquisas futuras, de fácil acesso e possível de ser enriquecido e complementado ao longo do processo de consolidação da IoT, agregando novos protocolos com suas respectivas vulnerabilidades.

4.1 Tipos de Segurança Implementados

A Tabela 4 apresenta um resumo dos modos de segurança implementados em cada protocolo, que foram descritos na Seção 3. Na tabela, AAA se refere a Autenticação, Autorização e Prestação de Contas, do inglês *Authentication, Authorization and Accountability* e a confidencialidade descreve os cifradores que o protocolo utiliza.

Table 4: Resumo dos modos de segurança implementados

Protocolo	Modos de segurança	AAA e Integridade	Confidencialidade
6loWPAN / 6TiSCH	Não implementa segurança	-	-
DTLS	*Record Protocol, Handshake Protocol, Alert Protocol, Change Cipher Spec*	HMAC-SHA1, HMAC-SHA256/ 384 e AEAD	Possui vários cifradores que podem ser selecionado
COAP	*No Sec, PreSharedKey, Raw Public Key* e *Certificate*	Lista de Raízes Confiáveis, Utiliza o DTLS	AES-CCM
MQTT	Utiliza o DTLS	Campo para o nome e senha utiliza DTLS	Utiliza DTLS

4.2 Tipos de Ataques Identificados

Vários ataques e vulnerabilidades foram identificados em cada protocolo. A Tabela 5 traz os ataques comuns em cada camada, na qual, caso tenha sido identificado no trabalho, o ataque é marcado. Em alguns casos, são colocadas observações, dado que certos ataques só ocorrem quando mecanismos de segurança disponíveis não são aplicados.

A camada física e de enlace, dada a abertura em que a comunicação sem fio apresenta, está suscetível a diversos ataques. Entretanto, também existem diversas soluções para tentar mitigar tal vulnerabilidade, porém, é intrínseco do ambiente que tais ataques possam ser aplicados. O *jamming*, por exemplo, pode ocorrer também na camada de enlace. Ao se transmitir informação sem criptografia em redes sem fio, qualquer indivíduo pode ter acesso a ela.

Na camada de rede foram identificados diversos ataques comuns dentro do processo de roteamento. Os ataques DoS ganham especial relevância no ambiente de IoT pois visam esgotar a bateria dos dispositivos. Aplicar o IPv6 nesse ambiente é também uma

Table 5: Ataques que podem ser explorados nos protocolos por camada

Camada	Protocolo	Aplicação																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Aplicação	CoAP																X	X**	
	MQTT																	X**	X**
Transporte	DTLS	X					X		X		X	X	X		X	X			
Rede	6loWPAN		X		X			X		X				X					
Física e Enlace	RFID e NFC		X*	X	X	X			X										

*Modos de segurança sem Criptografia. **CoAP, MQTT, estão vulneráveis a certos ataques somente quando não utilizam DTLS. 1- SSL Stripping, RC4, Problema de Usabilidade; 2- *Eavesdropping*; 3- *Relay*; 4- *Man-in-the-Midle*; 5- *Jamming* físico; 6- Injeção de Comandos; UDP *Flooding*; 7-*Spoofing*; 8- DoS; 9- Fragmentação; 10- CRIME, TIME, *Breach*, Parâmetro *Diffie-Hellman*; 11- BEAST, *Passing Oracle*, Problema de Implementação; 12- Roubo de Chaves do RSA, Certificado RSA; 13- *Realy Attack*; 14- *Handshake* Triplo, Renegociação; 15- Confusão de Hospedeiro Virtual; 16- Amplificação; 17- *Masquerading*; 18- *Trashing*;

tarefa complexa, que envolve a compressão e fragmentação e pode, consequentemente, dar espaço a ataques.

O DTLS foi colocado na camada de transporte, apesar do mesmo ser uma adaptação para segurança entre as camadas de aplicação e de transporte. Consideram-se, então, como vulnerabilidades da camada de transporte as vulnerabilidades do DTLS.

Dos protocolos da camada de aplicação, poucos possuem segurança embutida como padrão. A maioria se utiliza da segurança provida pelo DTLS. Cabe ao administrador da rede e desenvolvedores verificar a sensitividade dos dados e as restrições dos nós para implementar corretamente a segurança. Vale ressaltar que as ameaças desta camada são mais relacionadas ao software da aplicação do que aos protocolos em si.

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho, ao abordar aspectos relevantes voltados para segurança em IoT, trouxe uma visão macro da necessidade urgente de serem adotadas medidas para mitigar vulnerabilidades e ameaças, em sua maioria, do tipo DoS. Identificou-se as principais ameaças existentes e foram apresentadas algumas sugestões para evitá-las. Foi possível, então, agrupar as ameaças encontradas por camada de acordo com a atuação de cada protocolo. Como trabalhos futuros, um possível estudo, seria aplicar o mesmo método de pesquisa adotado neste artigo aos protocolos *Bluetooth*, BLE, IEEE 802.15.4, *Z-Wave*, *Wifi Direct* e LTE-A.

REFERENCES

[1] [n. d.]. RFID vs. NFC: What's the Difference? http://blog.atlasrfidstore.com/rfid-vs-nfc

[2] 2011. Coap Technology. Retrieved 15/05/2017 from http://coap.technology/

[3] 2011. IPv6 over the TSCH mode of IEEE 802.15.4e (6tisch). Retrieved 18/05/2017 from https://datatracker.ietf.org/wg/6tisch/charter/

[4] 2015. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. Retrieved 07/05/2017 from https://www.gartner.com/newsroom/id/3165317

[5] 2016. Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1. Retrieved 19/06/2017 from http://www.iso.org/iso/catalogue_detail.htm?csnumber=69466

[6] Nikolaos Alexiou, Stylianos Basagiannis, and Sophia Petridou. 2016. Formal security analysis of near field communication using model checking. *Computers & Security* 60 (2016), 1–14.

[7] Leonardo A Amaral, Fabiano P Hessel, Eduardo A Bezerra, Jerônimo C Corrêa, Oliver B Longhi, and Thiago FO Dias. 2011. eCloudRFID–A mobile software framework architecture for pervasive RFID-based applications. *Journal of Network and Computer Applications* 34, 3 (2011), 972–979.

[8] Sara Amendola, Rossella Lodato, Sabina Manzari, Cecilia Occhiuzzi, and Gaetano Marrocco. 2014. RFID technology for IoT-based personal healthcare in smart spaces. *IEEE Internet of things journal* 1, 2 (2014), 144–152.

[9] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. 2018. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications* 38 (2018), 8–27.

[10] Jordán Pascual Espada, Oscar Sanjuán Martínez, Juan Manuel Cueva Lovelle, B Cristina Pelayo G-Bustelo, Manuel Álvarez Álvarez, and Alejandro González García. 2011. Modeling architecture for collaborative virtual objects based on services. *Journal of Network and Computer Applications* 34, 5 (2011), 1634–1647.

[11] Roy Fisher and Gerhard Hancke. 2014. DTLS for lightweight secure data streaming in the internet of things. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*. IEEE, 585–590.

[12] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. 2015. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials* 17, 3 (2015), 1294–1312.

[13] Nurul Halimatul Asmak Ismail, Rosilah Hassan, and Khadijah WM Ghazali. 2012. A study on protocol stack in 6lowpan model. *Journal of Theoretical and Applied Information Technology* 41, 2 (2012), 220–229.

[14] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* 4, 5 (2017), 1125–1142.

[15] Anne H Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z Sheng. 2017. IoT middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal* 4, 1 (2017), 1–20.

[16] Kim Thuat Nguyen, Maryline Laurent, and Nouha Oualha. 2015. Survey on secure communication protocols for the Internet of Things. *Ad Hoc Networks* 32 (2015), 17–31.

[17] Ismael Peña-López et al. 2005. ITU Internet report 2005: the internet of things. (2005).

[18] Pavan Pongle and Gurunath Chavan. 2015. A survey: Attacks on RPL and 6LoWPAN in IoT. In *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 1–6.

[19] Reem Abdul Rahman and Babar Shah. 2016. Security analysis of IoT protocols: A focus in CoAP. In *Big Data and Smart City (ICBDSC), 2016 3rd MEC International Conference on*. IEEE, 1–7.

[20] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. 2015. *Summarizing known attacks on transport layer security (TLS) and datagram TLS (DTLS).* Technical Report.

[21] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. *The constrained application protocol (CoAP).* Technical Report.

[22] Shubhangi A Shinde, Pooja A Nimkar, Shubhangi P Singh, Vrushali D Salpe, and Yogesh R Jadhav. 2016. MQTT-message queuing telemetry transport protocol. *International Journal of Research* 3, 3 (2016), 240–244.

[23] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer networks* 76 (2015), 146–164.

[24] Meena Singh, MA Rajan, VL Shivraj, and P Balamuralidhar. 2015. Secure mqtt for internet of things (iot). In *Communication systems and network technologies (CSNT), 2015 fifth international conference on*. IEEE, 746–751.

[25] Saniya Vohra and Rohit Srivastava. 2015. A survey on Techniques for Securing 6LoWPAN. In *Communication Systems and Network Technologies (CSNT), 2015*

Fifth International Conference on. IEEE, 643–647.

[26] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. 2017. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal* 4, 5 (2017), 1250–1258.

[27] Azam Zavvari and Ahmed Patel. 2012. Critical evaluation of RFID security protocols. *International Journal of Information Security and Privacy (IJISP)* 6, 3 (2012), 56–74.