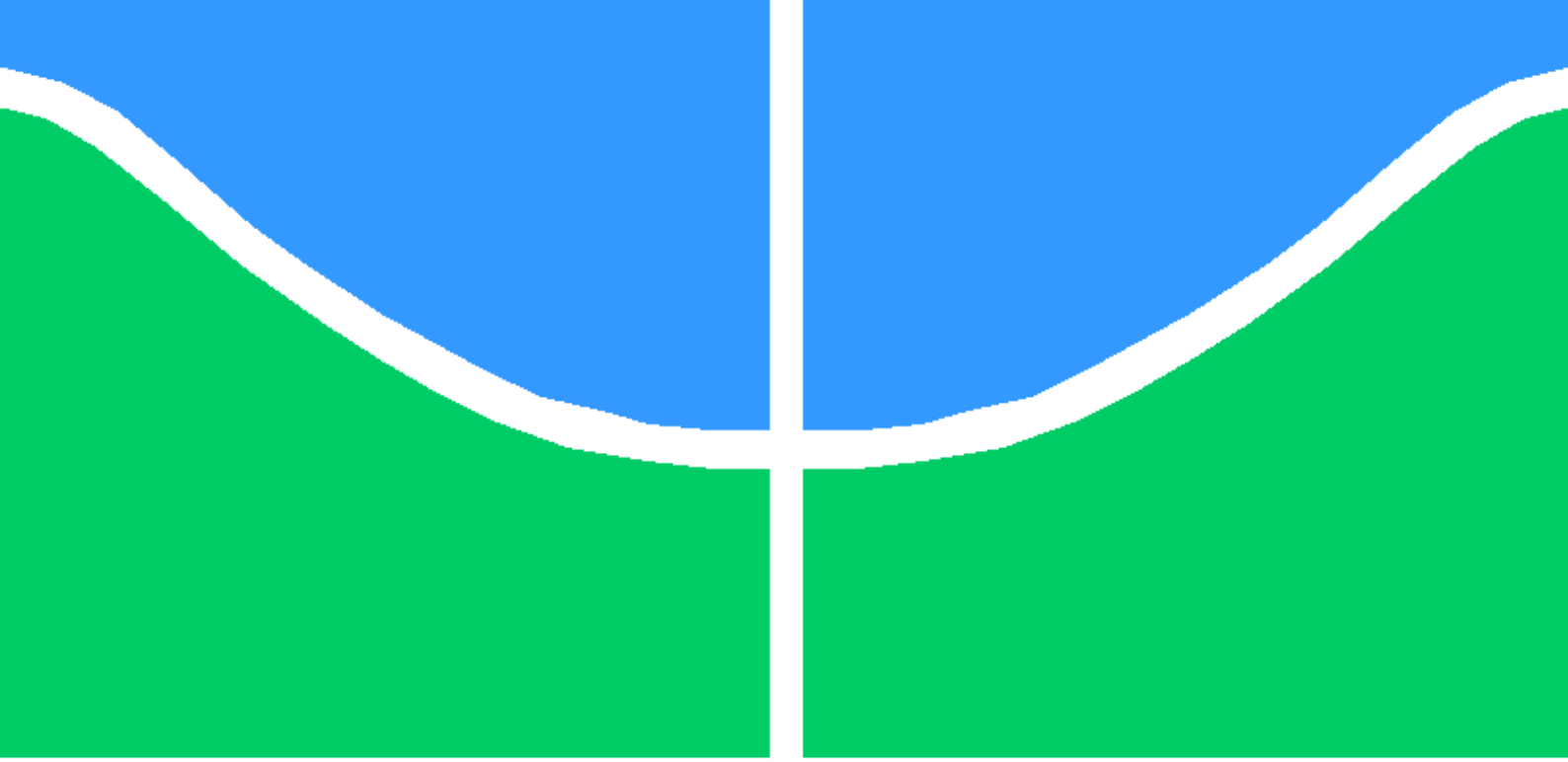

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Modelo Hierárquico de Middlewares com Aplicação de Edge Computing para Redes IoT

João Tribouillet Marcial de Menezes

Pedro Henrique Lira da Costa

Brasília, Julho de 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Modelo Hierárquico de Middlewares com Aplicação de Edge Computing para Redes IoT

João Tribouillet Marcial de Menezes

Pedro Henrique Lira da Costa

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. PhD. Rafael Timóteo de Sousa Júnior, _____
ENE/UnB
Orientador

MSc. Francisco Lopes de Caldas Filho, _____
ENE/UnB
Examinador Externo

Prof. Msc. Valerio Aymoré Martins _____
Examinador

Nada na vida é para ser temido, é apenas para ser entendido. Agora é a hora de entendermos mais para que possamos temer menos.

Traduzido, Marie Curie

Agradecimentos

Agradeço primeiramente a minha mãe Elaine Marcial, que sempre me apoiou durante toda a minha jornada de estudos e de vida, não sendo diferente na presente pesquisa, a minha família, ao Laboratório UIoT e a FAP-DF, pelo apoio durante o desenvolvimento desse projeto, aos meus amigos e parceiros, pelo companheirismo e apoio ao longo de minha história, em especial a Dayanne Fernandes durante todo o tempo de participação do projeto do UIoT, sendo ela uma fonte de inspiração e de conhecimentos além de todo o apoio ao longo do desenvolver do projeto, e ao Pedro Henrique durante todo o período de graduação da Universidade de Brasília, sendo ele meu maior companheiro durante essa jornada com inúmeros dias e madrugadas de trabalhos e estudos sendo, assim, imprescindível para vários objetivos alcançados. Por fim agradeço ao professor orientador Rafael Timóteo pelo apoio dado para a realização desse projeto.

João Tribouillet Marcial de Menezes

Agradeço a Deus, a minha família por possibilitar e apoiar a realização dos meus estudos, aos meus amigos, a equipe do laboratório UIoT, em especial a Dayanne Fernandes, ao meu Professor orientador Rafael Timóteo e a todos que nos orientaram nesse trabalho. Agradeço a PhD. Elaine Coutinho Marcial pelos ensinamentos. Por fim, agradeço meu grande amigo João Tribouillet por todo empenho e dedicação ao nosso trabalho.

Pedro Henrique Lira da Costa

Resumo

O crescimento em larga escala da geração e tráfego de dados na Internet possibilitou o desenvolvimento de análises complexas em diversos contextos. Com tal avanço, apareceram desafios na coleta, processamento, persistência e apresentação de dados para gerar informações.

Os consequentes desafios ligados ao alto consumo de banda, tempo de resposta e recursos para processamento são particularmente interessantes quando se considera esse fenômeno da grande quantidade de dados em redes de Internet das Coisas (IoT), haja vista a sempre crescente quantidade de dispositivos, a variabilidade dos dados e a velocidade de geração e consumo de dados nesse tipo de rede.

Nesse sentido, este trabalho propõe uma estrutura descentralizada para interferir no processamento e tráfego de redes IoT, com aplicação de regras definíveis de coleta, sumarização e repasse de dados, durante o fluxo na IoT, de forma a responder aos citados desafios, utilizando a capacidade de processamento de componentes que se encontram na borda da rede para aplicar tais regras e reduzir o tráfego gerado, bem como racionalizar o processamento central.

Para tanto, foram concebidos os componentes da arquitetura hierárquica de rede com utilização de *Edge Computing* e para a validação dessa arquitetura, forma desenvolvidas e detalhadas ferramentas que permitiram validar a alternativa proposta para superar os desafios da geração e tráfego de dados em IoT.

Abstract

The large-scale growth of data generation and traffic on the Internet has enabled the development of complex analyzes in various contexts. With such advancement, challenges appeared in the collection, processing, persistence and presentation of data to generate information.

The consequent challenges associated with high bandwidth consumption, response time and resources for processing are particularly interesting when considering this phenomenon of the large amount of data in the Internet of Things (IoT), given the ever increasing number of devices, the variability of data and the speed of generation and consumption of data in this type of network.

In this sense, this work proposes a decentralized structure to interfere in the processing and traffic of IoT networks, with application of definable rules of data collection, summarization and data transfer during the IoT flow, in order to respond to the aforementioned challenges, using the capacity of processing components that are on the edge of the network to apply such rules and reduce the traffic generated, as well as rationalize the central processing.

For this, the components of the network hierarchical architecture using textit Edge Computing and for the validation of this architecture, were developed and detailed tools that allowed to validate the proposed alternative to overcome the challenges of generation and data traffic in IoT.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Acrônimos

API	Application Programming Interface
UnB	Universidade de Brasília
IoT	Internet of Things
HTTP	Hypertext Transfer Protocol
VPS	Virtual Private Server
CPU	Central Process Unit
VCC	Common Collector Voltage
GND	Ground
QoE	Quality of Experience
RAM	Random Access Memory
kB	kilobyte
MB	Megabyte
GB	Gibabyte
M2M	Machine to Machine
DIMS	Data Interface Management System
IPv	Internet Protocol version
CDN	Content Delivery Network
SD-WAN	Software Defined Wide-Area Network
CAGR	Compound Annual Growth Rate
PC	Personal Computer
IP	Internet Protocol
HD	High Definition
VoD	Video on DEmand
RFiD	Radio Frequency Identification
Ubicomp	Ubiquitous Computing
WSN	Wireless Sensor Network
A/D Converters	Analog Digital Converters
MAC	Medium Access Control
WSN	Wireless Sensor Network
SOA	Service Oriented Architecture
OSWA	Open Sensor Web Architecture
RAN	Radio-Access Network
MEC	Mobile Edge Computing
I/O	Input/Output
DHT11	Digital Humidity Tempeture Sensor
GPIO	General Purpouse Input/Output

1 Introdução

O crescente desenvolvimento de tecnologias permite a obtenção e controle de dados antes inexplorados proporcionando um melhor controle de variados sistemas presentes na sociedade, tais como transporte, cuidados de saúde, monitoramento de ambientes e de pessoas, por exemplo. Com o avanço tecnológico e reconhecida a necessidade humana do gerenciamento automático de dados, proporcionou-se a execução de coleta de métricas que antes eram de difícil acesso. Esse avanço também resultou em um aumento do volume de dados já coletados, seja pelo aumento da capacidade de persistência de dados, da velocidade de coleta ou da distribuição de geradores de dados.

Essa evolução crescente da informação digital possibilita, portanto, uma melhor gestão de aspectos humanos, sociais ou tecnológicos. A informação presente em dados extraídos de fontes variadas são utilizadas em aplicações que, muitas vezes, necessitam de um pré-processamento para a efetiva obtenção da informação útil para sua atuação.

Há diversas estimativas relacionadas ao crescimento do volume de dados e informações trafegando na rede, em especial com a expansão da IoT.

O Cisco *Visual Networking Index* - VNI (Cisco, 2019), presente no anexo A deste trabalho, é um relatório que apresenta previsões para o tráfego mundial total da Internet e outras questões ligadas a temática como previsões sobre o número de dispositivos conectados à Internet de forma quantitativa e sempre crescentes. O estudo expõe importantes pontos de análise como mudanças relativas ao aumento da variedade de dispositivos e conexões, que mostra uma estimativa de mais de 25 bilhões de dispositivos no mundo e destes, mais da metade correspondem a conexões M2M[1]. Isso reflete no crescimento da quantidade de dispositivos IoT presentes na rede em âmbito mundial e no crescimento do tráfego de aplicações presentes na Internet. Neste contexto, a estimativa é que em 2022 o tráfego global da Internet seja de 150.000 GB por segundo. Diante de tal cenário, estudos relativos a mudanças para suportar o tráfego previsto são indispensáveis.

Há possíveis curingas (*wild cards*[2]) que necessitam ser monitorados, visto que a abordagem da Cisco para previsão de tráfego IP foi considerada conservadora. Inovações emergentes em arquiteturas de rede, implantações de dispositivos/conexões e adoção/uso de aplicativos possuem o potencial de aumentar as perspectivas de volume de tráfego, formas e características. São eles:

- Transformação de Arquitetura de Rede / Infraestrutura, com as *edge networking* e a implementação do 5G, que representa a nova geração da rede móvel com uma maior taxa de transferência de dados.

[1] M2M - *Machine to Machine, que representam conexões diretas entre dois dispositivos*

[2] Curingas - referem-se a grandes surpresas, difíceis de serem antecipadas ou entendidas, possuem pequena probabilidade de ocorrência, são de grande impacto, e geralmente surpreendem a todos, em parte porque se materializam muito rapidamente, tão rapidamente que sistemas sociais não podem efetivamente respondê-los.

- Deslocamentos de dispositivo/conexão, com o domínio dos *smartphones* como o "*hub* de comunicação" para mídias sociais, consumo de vídeo, rastreamento de aplicativos de IoT/digitalização e para comunicação tradicional em voz. Outro ponto seria o impacto de conexões e aplicativos de IoT criando novas demandas e requisitos de rede.

- Adoção e uso de aplicativos, como os jogos na Internet, a realidade virtual e ampliada e vigilância por vídeo, em função da proliferação de câmeras conectadas a internet.

Dado os pontos expostos, o uso dos canais na internet para a transferência de dados se mostra crescente tanto em trafego quanto em quantidade de dispositivos conectados a rede. Nesse contexto, Internet das Coisas (IoT) se mostra relevante, pois coleta os mais variados dados importantes para controle de sistemas e auxílio na tomada de decisões. O uso de dispositivos IoT contribui, também, com o aumento do volume de informação, que deverá transitar na Internet e ser armazenado. Espera-se que esse volume continue a crescer de forma exponencial, como pode ser evidenciado na Figura 1.1. Logo o uso desses dispositivos resulta em uma quantidade massiva de dados transitando na Internet, pois são encarregados apenas de coletar e mandar os dados para um Middleware ou *cloud* [3].

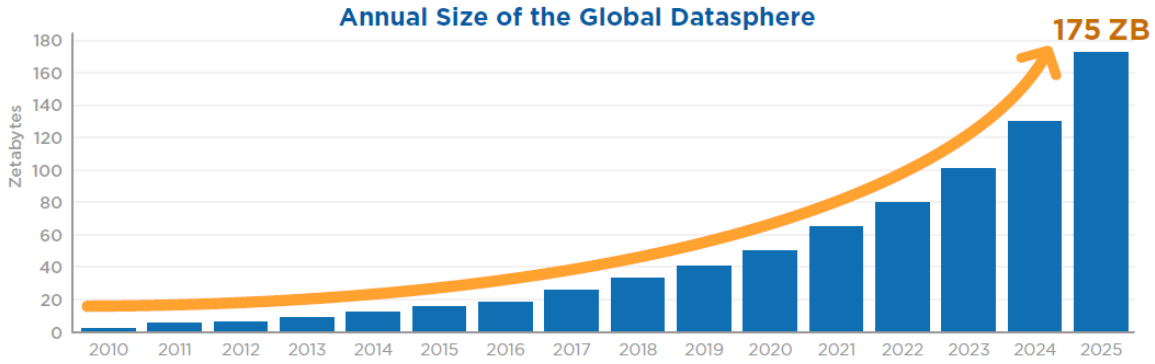

Figura 1.1 – Crescimento da esfera de dados mundial
Fonte: IDC [19].

O aumento da complexidade dos sistemas deixou mais clara a necessidade de processamento automático dos dados. O processamento automático pode, além de gerar informação, ser atrelado a uma atuação também automática sobre o sistema. Portanto, a coleta, o controle e o processamento dessa imensa quantidade e variedade de dados se tornou imprescindível para o gerenciamento dos sistemas e atuadores no mundo. O conceito de IoT permite, com mais facilidade, a interoperabilidade de sistemas ou sub-sistemas, sendo um ótimo candidato para tal tarefa.

O processamento de dados é importante para obter informações úteis sobre as métricas cruas coletadas, que muitas vezes é feito a partir de uma base de dados extensa na *cloud* ou por outras entidades que possuem acesso aos dados que foram persistidos nela, sendo que neste caso, há um

[3] CLOUD - ***Computing*** *resources that is* ***Location*** *independent, can be accessed via* ***Online*** *means, used as an* ***Utility*** *and is available on* ***Demand***

gasto de banda para a obtenção dos dados necessária para o processamento. Este trabalho tem como interesse buscar uma solução mais otimizada que a arquitetura de *cloud*, visto que para a persistência, processamento e resposta/atuação, esta não se mostra uma boa alternativa para todas as aplicações. A aglomeração de dados em um único ponto torna o processamento mais custoso e mais lento, além de adicionar uma latência para respostas às requisições de aplicações ou clientes (Juyong Lee e Jihoon Lee [13, 25]).

1.1 Justificativa

Dados os desafios do crescente volume de dados que trafegam e são persistidos pela Internet, é possível perceber uma necessidade de mudança na arquitetura de *cloud* central, visto que todo o tráfego deverá transitar no núcleo da Internet.

Soluções para os problemas apresentados relativos a arquitetura com uma *cloud* central estão em estudo e ainda não existe uma solidez e unanimidade no que tange a solução perfeita para os cenários do uso abrangente a coleta de dados e envio de dados para atender diferentes aplicações. Nesse contexto, a presente pesquisa busca utilizar recursos associados ao conceito de *Edge Computing*[4], como solução para os referidos problemas.

A aplicação de uma arquitetura que utiliza *Edge Computing* se mostra uma boa alternativa. A escalabilidade dos serviços já existentes ou de serviços futuros depende de uma arquitetura capaz de atender a demanda na borda da Internet.

Middlewares intermediários ou de borda da rede podem ser responsáveis por lidar e persistir a maior quantidade de dados brutos vindos de dispositivos, evitando competição com o tráfego já existente na rede. A distribuição geográfica desses *Middlewares* poderia repartir melhor a persistência de dados coletados próximos a eles, tirando a necessidade da transição desses dados por um *backbone* da Internet. É de tratamento dessa pesquisa o tópico de transferência de dados para entidades fora da extrema borda da Internet, tornando possível o envio de dados por requisições pontuais ou por cadastro de serviços de sumarização.

Há também a possibilidade de obtenção de dados e informações pela implantação de regras. A partir disso, um fluxo controlado existiria entre as entidades que compõem a arquitetura, sendo que estas permitem sumarização dos dados. Com isso, uma quantidade de dados transitados pela rede seria reduzido, sem perda de informação, pois as entidades conseguem requisitar informação útil e aplicável para o seu escopo de gerência.

1.2 Objetivos

Essa pesquisa apresenta os objetivos gerais e específicos descritos a seguir.

[4] Utilização de processamento na borda da rede que permite diminuição da latência e menor uso de banda, vide seção 2.4

1.2.1 Objetivo Geral

- Propor um modelo hierárquico com aplicação de recursos associados ao conceito de *Edge Computing* para redes IoT por meio da criação de um *Middleware* genérico capaz de atuar ao longo da rede.

1.2.2 Objetivos Específicos

- Verificar se a arquitetura proposta apresenta melhores resultados que a arquitetura em *cloud* centralizada.
- Desenvolver um sistema de atuação/alerta genérico.
- Desenvolver dispositivos geradores de dados para validação da arquitetura proposta.

1.3 Hipótese

A implantação de uma rede IoT com *Middlewares* geo-distribuídos de forma hierarquizada é capaz de diminuir consideravelmente a quantidade de dados transitados pelo núcleo da Internet e permitir sumarização de dados a partir do processamento que explora o recursos associados ao conceito de *Edge Computing*.

1.4 Metodologia

A pesquisa apresentada neste documento é de caráter experimental, começando com a proposição de um modelo hierárquico com aplicação de recursos associados ao conceito de *Edge Computing* para redes IoT. Foi assim criado um *Middleware* genérico capaz de atuar ao longo da rede.

Com tal *Middleware* desenvolvido na forma de um protótipo, foram executados testes de validação de modo a comparar a arquitetura proposta com uma arquitetura em *cloud*.

Os resultados dos dois testes foram coletados, analisados e comparados. Para esse fim, utilizou-se as métricas uso de CPU [5], tráfego de rede e quantidade de dados produzidos e transitados.

1.4.1 Restrições da pesquisa

- A arquitetura proposta e implementada não leva em consideração segurança na comunicação, nem entre dispositivos e nem entre os *Midllewares* hierarquizados.
- O *Middleware* desenvolvido prevê, mas não implementa automatização no processamento, no processo de decisão e no cadastro de regras em qualquer *Middlewares*.
- Na implementação da arquitetura proposta, é premissa que os *Middlewares* já são conectados via rede.

[5] CPU - *Central Process Unit*.

1.5 Apresentação do trabalho

Incluindo este capítulo, o trabalho é organizado da seguinte forma:

1. Capítulo 1: Apresentação da pesquisa realizada, incluindo problemática e apontamento de objetivos e hipóteses. Apresentação dos desafios presentes e futuros no âmbito de dados na Internet, tendo em vista o tópico de Internet das Coisas.
2. Capítulo 2: Apresentação e discussão dos conceitos, desafios e tecnologias envolvidos na elaboração da problemática da pesquisa.
3. Capítulo 3: Detalhamento da metodologia e ferramentas utilizadas além da arquitetação do modelo de implementação da solução.
4. Capítulo 4: Exposição e detalhamento de todos os artefatos produzidos bem como relato e análise de testes executados.
5. Capítulo 5: Síntese e discussão dos resultados obtidos e abertura para trabalhos de melhoria nos temas e ferramentas apresentados.

2 Fundamentação Teórica

Neste capítulo será apresentada a revisão da literatura realizada para os temas: o conceito de *Internet of Things* (IoT), sua evolução, os principais componentes de uma IoT e a arquitetura hierárquica com *edge computing* para a IoT.

2.1 Internet of Things

O crescimento da existência de objetos interconectados em uma infraestrutura de rede de comunicação moderna ao nosso redor, sejam eles sensores ou outros dispositivos que coletam e trocam dados diferentes entre si, marca uma nova era regida por um novo paradigma o da *Internet of Things* que, rapidamente, será aplicada em diversos cenários do cotidiano da sociedade [9, 10, 29]. Esse novo mundo produzirá um volume de dados da ordem de *Zeta Bytes* (conforme indicado na Figura 1.1) que precisarão ser armazenados, processados e apresentados de forma transparente, eficiente e de fácil interpretação tornado a IoT uma importante função na vida das pessoas [9, 29].

O termo *Internet of Things* foi cunhado por Kevin Ashton em 1999, cuja concepção inicial descrevia a relação entre uma "nova ideia", à época, a do RFID[1] na gestão de cadeias de suprimento da P&G[2] [2, 24]. Durante algum tempo a IoT foi considerada como uma mera extensão da identificação por radiofrequência [24].

Segundo o *Cluster of European Reseach Project on the Internet of Things* [24], os trabalhos listados a seguir referem-se aos primeiros de interesse geral sobre a IoT, marcando o início dessa nova era para o comércio e a indústria.

- "The Internet of Things", por Sean Dodson, The Guardian, Outubro de 2003.
- "Toward a Global Internet of Things", por Steve Meloan, Sun Microsystems, Novembro de 2003.
- "A Machine-to-Machine Internet of Things", Business Week, Abril de 2004.
- "The Internet of Things", por Neil Gershenfeld, Raffi Krikorian and Danny Cohen, Scientific American Magazine, Outubro de 2004.
- "The Internet of Things: Start-ups jump into next big thing: tiny networked chips", por Robert Weisman, The Boston Globe, Outubro de 2004.

Entretanto, somente em 2005, quando a União Internacional de Telecomunicações (UTI) em [11] publicou o primeiro relatório sobre "Internet das Coisas", foi que o tema ganhou destaque ao apresentar uma abordagem mais abrangente e holística sobre o tema ao tempo em que sugeria que a:

[1] RFID - *Radio Frequency Identification.*
[2] P&G - *Procter & Gamble Company.*

> (...) IoT se conectará aos objetos do mundo de maneira sensorial e inteligente por meio da combinação de desenvolvimentos tecnológicos na identificação de itens, sensores e redes de sensores sem fio, sistemas embutidos ("coisas pensantes") e nanotecnologia ("encolhendo coisas") [11][tradução livre dos autores].

O termo "*Internet of Things*" sofreu evolução ao longo do tempo ao ter sua aplicação ampliada para áreas como saúde, educação, serviços públicos, transporte, dentre outros. Mesmo tendo sua ênfase em "*things*", a partir da evolução tecnológica essa visão vem sendo alterada e o foco da IoT passou a ser a captura e armazenamento de dados, visualização e produção de informação capturadas sem a intervenção humana. Essa rede de objetos interconectados, que tanto extraem dados do ambiente, por meio de sensores, quanto interage com o mundo físico (atuação/comando/controle) por meio de padrões existentes na Internet, produz informação que subsidia o fornecimento de serviços, criando um ambiente inteligente [9, 2].

Corroborando com essa visão, o *RFID group* define Internet das Coisas como: *"A rede mundial de objetos interconectados, endereçável exclusivamente com base em protocolos de comunicação padrão"*‘ [9].

De acordo com o Cluster of European Research Project on the Internet of Things em [24]:

> "Objetos" são participantes ativos em negócios, informações e processos sociais onde eles são capazes de interagir e se comunicar entre si e com o ambiente, trocando dados e informações sobre o meio ambiente, enquanto reage autonomamente aos eventos do mundo real/físico e influenciá-lo, executando processos que desencadeiam ações e criar serviços com ou sem intervenção humana direta [24] (tradução livre dos autores).

Nesse contexto, IoT pode ser entendida como um conceito que compreende a interconectividade entre objetos físicos entre si ou entre eles à Internet, endereçável exclusivamente com base em protocolos de comunicação padrão, sem a intervenção humana, para a criação de um ambiente inteligente que, por meio de um sensoriamento contínuo (monitoramento), fornece captura, armazenamento, analise, visualização e entrega ao cliente dos dados produzidos, bem como interage com o mundo físico (atuação/comando/controle) por meio de padrões existentes na internet, produzindo e disponibilizando informação transparente para a tomada de decisão e fornecimento de serviços [9, 10, 2, 24].

Esses ambientes inteligentes para IoT podem ser entendidos como:

> [...] a interconexão de dispositivos sensores e atuadores, proporcionando capacidade de compartilhar informações entre plataformas por meio de uma estrutura unificada, desenvolvendo e permitindo aplicações inovadoras. Isto é conseguido com detecção onipresente, análise de dados e representação de informações com a computação em nuvem como estrutura unificadora [9] (tradução livre dos autores).

A IoT foi impulsionada pela prevalência de dispositivos habilitados com tecnologia sem fio aberta, como *bluetooth*, identificação por radiofrequência (RFID), Wi-Fi e serviços de dados telefônicos, em especial os *smartphones* bem como pela incorporação de sensores e atuadores e quaisquer outros dispositivos incorporados que poderão ser conectados e participar de troca de informações.

Esses dispositivos são, ainda, capazes de interagir uns com os outros para o atingimento de objetivos comuns [9, 3, 29].

2.2 Componentes de uma IoT

A internet das coisas é formada por diversos componentes que interagem entre si e permitem que essa rede funcione. Gubbi *et al.* em [9] propõem uma taxonomia que ajuda a definir os componentes para a Internet of things, sob uma perspectiva de alto nível, são eles: *Hardware*, *Middleware* e *Presentation* que serão desenvolvidos nas subseções que se seguem.

2.2.1 Hardware

No campo do *hardware*, um dos principais componentes que viabilizaram a evolução da IoT foi a tecnologia RFID. A proliferação das etiquetas RFID impulsionou o crescimento de vários conceitos que integram o mundo físico com o virtual e contribuiu com os avanços no campo da comunicação que permitiram o *design* de microchips para comunicação de dados sem fio. Esses dispositivos auxiliam na identificação automática de qualquer coisa a que estejam ligados por meio de um código de barras eletrônico e funcionam sem a necessidade de serem alimentadas por baterias. O RFID possui múltiplas aplicações com no gerenciamento de cadeias de suprimentos e varejo, no monitoramento de cargas, nos transportes, em aplicativos de controle de acesso, nos cartões bancários, dentre outros, viabilizando a IoT [9, 3, 27].

Outro componente importante no campo dos *hardwares*, que viabilizaram a IoT, são as redes de sensores sem fio (WSN[3]), que também podem cooperar com as etiquetas RFID. Em geral, um nó (*hardware* principal da WSN) contém interfaces de sensores, unidades de processamento, unidades de transceptor e fonte de alimentação. Em geral, são compostos de múltiplos conversores analógico para digital (A/D *converters* [4]) para a interface do sensor. No caso dos nós de sensores mais modernos, esses têm a capacidade de se comunicar usando uma banda de frequência, tornando-os mais versáteis [9, 1].

Essas redes, compostas por um grande número de sensores inteligentes, permitem a coleta, processamento, análise e disseminação de informações valiosas, reunidas em uma variedade de ambientes agem como uma ponte entre o mundo físico e o digital. As aplicações são vastas desde o monitoramento ambiental (de temperatura, umidade, pressão, níveis de ruído etc.), as aplicações de saúde, sistemas de transporte inteligentes, eficiência energética, aplicações militares e de segurança e monitoramento de plantas industriais, por exemplo. Os avanços tecnológicos no campo dos circuitos integrados de baixa potência e comunicação sem fio resultaram na proliferação desses dispositivos miniaturizados cada vez mais eficientes, de baixo custo e consumo de energia que ampliaram o uso em aplicações de sensoriamento remoto e viabilizam a IoT [9, 3, 1].

As pilhas de comunicação das redes de sensores sem fio representam outro componente importante ao possibilitarem que os nós sejam implantando de maneira *ad-hoc* para a maioria das aplicações. Ao se pensar em escalabilidade e longevidade de uma rede torna-se importante projetar

[3] WSN - *Wireless Sensor Network.*
[4] A/D converters - Analog Digital Converters.

uma topologia, um roteamento e uma camada MAC[5] apropriados. Na transmissão dos dados é necessário que os nós em uma WSN se comuniquem entre si e a pilha de comunicação no nó deve possibilitar a interação com o mundo externo por meio da internet, atuando como um *gateway* para a subrede WSN e a Internet [3, 8].

2.2.2 Middleware

Quanto ao *Middleware*, o segundo componente da taxonomia proposta por Gubbi *et al.* [9], representa uma camada de *software* composta por um conjunto de subcamadas interpostas entre os níveis tecnológico e de aplicação. Sendo assim, é um mecanismo que combina uma infraestrutura cibernética com uma arquitetura orientada a serviços (SOA[6]), que permite a decomposição de sistemas complexos e monolíticos em aplicações mais simples, tornando suas vantagens reconhecidas pela maioria dos estudos sobre soluções de *middleware* para IoT. As redes de sensores para fornecer acesso a recursos de sensores heterogêneos de maneira independente da implantação também são destaque. Além de armazenar os dados, promove computação para a análise dos dados armazenados fornecendo aplicações especificamente habilitadas pelas infraestruturas de IoT [9, 3, 8].

Segundo Atzori *et al.* em [8]:

> O *middleware* está ganhando cada vez mais importância nos últimos anos devido ao seu papel principal na simplificação do desenvolvimento de novos serviços e na integração de tecnologias legadas em novas. Isso isenta o programador do conhecimento exato do conjunto variado de tecnologias adotadas por camadas inferiores. [8](tradução livre dos autores).

Segundo Atzori *et al.* em [3], ainda não há uma arquitetura em camadas aceita. Sendo assim, as soluções enfrentam os mesmos problemas relacionados às funcionalidades dos dispositivos e as capacidades de comunicação. Esses objetivos comuns sugerem um modelo de *middleware* da IoT que procura abranger todas as funcionalidades descritas em trabalhos passados, apresentado na Figura 2.1.

[5] MAC - *Medium Access Control*
[6] SOA - *Service Oriented Architecture*

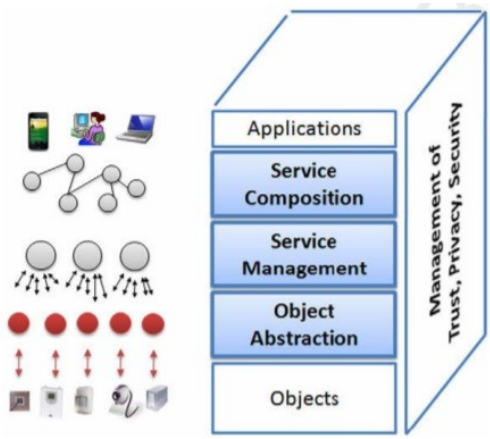

Figura 2.1 – SOA-based architecture for the IoT *middleware*
Fonte: L. Atzori, A. Iera e G. Morabito [3].

.

Os WSN *Middlewares* trabalham com o isolamento de recursos utilizados por várias aplicações. O desenvolvimento de aplicativos de sensores irá demandar um *middleware* independentemente da plataforma, que apresente uma arquitetura web de sensor aberto (OSWA[7]). [3, 21]. Segundo Atzori *et al.* em [3]:

> A OSWA é construída sobre um conjunto uniforme de operações e representações de dados padrão, conforme definido no Método de Capacitação da Web do Sensor (SWE) pelo Open Geospatial Consortium (OGC) [3](tradução livre dos autores).

2.2.3 Presentation

É por meio da visualização que a interação do usuário com o ambiente em aplicações IoT é permitida. O uso de *tablets*, *smatphones* e outros dispositivos que possuem tela sensível ao toque tornou essa interação intuitiva, permitindo que, mesmo pessoas leigas possam se beneficiar da revolução que a IoT trará. Para tanto, torna-se necessário a criação de uma visualização atraente e fácil de entender, o que se tornará ainda mais fácil com a chegada das telas 3D. Destaca-se que a extração de informação relevante advindas dos dados brutos não é trivial, englobando a detecção dos eventos, visualização dos dados brutos e modelados, que devem ser apresentados segundo as necessidades do usuário final [9].

As aplicações estão no topo da arquitetura SOA para a IoT *Middleware* (Figura 2.1) possibilitando a exploração de todas as funcionalidades do sistema para o usuário final. Mesmo não fazendo parte do *middleware*, explora as funcionalidades de cada camada [3].

Há diversas categorizações propostas para as aplicações disponíveis para IoT. Atzori *et al.* em [3] propõe quatro domínios: Transporte e logística; Cuidados de saúde, Ambientes inteligentes (casas, escritórios, plantas industriais) e Social e pessoal. Essa classificação é muito semelhante a de Gubbi *et al.* em [9], que também as categoriza em quatro domínios: Pessoal e Residencial;

[7] OSWA - Open Sensor Web Architecture

Empreendedor; Utilitários; e Celular. Ainda seguindo esses autores, existe um cruzamento entre aplicações e o de dados entre domínios. Em uma IoT Pessoal e Doméstica são produzidos dados que podem ser utilizados por empresas para otimização de serviços, como, por exemplo, medições do consumo de água, e tais empresas pode otimizar a oferta e demanda no Utilitário IoT.

A potencialidade das aplicações que utilizam de IoT se mostra enorme. Os domínios e os ambiente nos quais essas aplicações poderiam melhorar a qualidade de vida das pessoas são muitos, entretanto hoje em dia esses ambientes estão equipados com objetos com inteligência primitiva, na maioria das vezes sem qualquer capacidade de comunicação. Dar a esses objetos a possibilidade de se comunicar uns com os outros e elaborar as informações percebidas a partir de seu entorno implica em ter diferentes ambientes onde uma ampla gama de aplicações pode ser implantada.

2.3 Arquitetura Hierárquica com *Edge Computing* em Redes IoT

No campo do "*hierarchical edge architecture*", Tong *et al.* em [25] argumenta que uma arquitetura hierárquica de *cloud* com processamento na borda e não na internet, atende requisições com maior eficiência e rapidez, pois permite a agregação das cargas de pico dos usuários móveis em diferentes servidores em nuvem. Tal arquitetura busca a utilização eficiente dos recursos de nuvem, por meio de um algoritmo de posicionamento de carga de trabalho que decide em que programas móveis de servidores de nuvem na borda deverão ser colocados e a quantidade de capacidade computacional que deve ser provisionada para executar cada programa.

Justificam a necessidade de redesenhar a arquitetura em *cloud* e o desenvolvimento desse tipo de arquitetura para enfrentarem o desafio da computação móvel dada a crescente complexidade das aplicações móveis e as capacidades locais limitadas desses dispositivos. Mostram as vantagens desse tipo de arquitetura hierárquica na borda e a eficácia e escalabilidade do algoritmo proposto. Advogam que:

> O desempenho da computação móvel seria significativamente melhor aproveitando a *cloud computing* e migrando cargas de trabalho móveis para execução remota na *cloud*. Para tanto, propuseram a implantação de servidores em nuvem na borda da rede e projetaram a *cloud* de borda como uma hierarquia em árvore de servidores distribuídos geograficamente, de forma a utilizar eficientemente recursos em nuvem para atender às cargas de pico dos usuários móveis. [25](tradução livre dos autores).

O problema já havia sido detectando anteriormente nos trabalhos de Satyanarayana *et al.* em [22] e Verbelen *et al.* em [26] que desenvolveram proposta de implantação de uma camada de *cloud* intermediária, chamadas de *cloudlets*, que funcionariam na borda da rede. Essa camada ajudaria a reduzir a latência de acesso a essa nuvem, entretanto a *cloudlet* não atenderia a demanda dos usuários ou iria disponibilizar recursos que não seriam utilizados em horários fora de pico e prejudicaria a eficiência da utilização de recursos em nuvem [25].

No caso da proposta de Tong *et al.* em [25], a ideia seria atender às cargas de pico que excedem as capacidades de camadas inferiores de servidores de nuvem na borda para outros servidores em camadas mais altas na hierarquia, resultando no atendimento a maiores quantidades de carga de

pico com mesma quantidade de capacidades computacionais provisionadas na borda. Também garante a utilização eficiente dos recursos de computação em diferentes camadas de servidores de nuvem em borda para melhor atender nas cargas de pico.

A expansão da IoT, resultado da proliferação de milhares de sensores e dispositivos conectados, resulta na produção contínua de dados e na troca de mensagens por meio de redes complexas que suportam comunicação máquina a máquina, além de monitorarem e controlarem infraestruturas críticas. Entretanto, a maioria dos dispositivos IoT possuem potência limitada e necessitam equilibrar o consumo de energia de forma a prolongar a vida útil dos dispositivos. Para tanto, torna-se necessário agendar a computação para dispositivos com maior capacidade e recursos computacionais [29].

Na computação em *cloud* convencional, todos os dados devem ser carregados em servidores centralizados e, após seu processamento, os resultados precisam ser enviados de volta para os sensores e dispositivos, sendo a velocidade de transmissão de dados afetadas pelo tráfego de rede. Esse processo cria uma grande pressão e congestionamento de recurso na rede, aumentando os custos de consumo de rede. Se o processamento de dados em nós de computação ocorrerem em nós de computação em distâncias menores até o usuário haverá a redução do tempo de transmissão [29].

Para mitigação desse problema, que deverá piorar com o aumento do volume dos dados trafegados em rede, torna-se necessário a formulação de uma estratégia com o objetivo específico de reduzir os custos de transmissão de dados, de largura de banda e recursos. Além disso, o desempenho da rede irá piorar. Nesse contexto, emerge como um novo paradigma a computação de borda da rede, próximo aos usuários finais, que melhora o desempenho de redes IoT [29].

Segundo Yu *et al.* em [29] a computação em borda de redes IoT representa uma nova arquitetura distribuída que engloba a computação e o armazenamento de dados próximos ao usuário, por conseguinte:

> (...) reduz significativamente a latência na troca de mensagens e transmissão entre os servidores *edge/cloudlet* e os usuários finais; pode equilibrar o tráfego de rede e evitar os picos de tráfego nas redes IoT; e reduzir os tempos de resposta das aplicações de IoT em tempo real em comparação aos serviços tradicionais de nuvem; estender a vida útil dos nós individuais ao transferir computação e sobrecarga de comunicação de nós com fornecimento limitado de bateria para nós com recursos de energia significativos [29].

Também reduz significativamente os requisitos de largura de banda de rede centralizada e reduz a latência de transmissão durante a computação ou armazenamento de dados em IoT. Uma possível forma de redução do tráfego, da pressão computacional que ocorrem na *cloud* centralizada e dos tempos de respostas de aplicações IoT seria por meio da distribuição de nós de computação implantados na borda.

Essa nova arquitetura hierárquica com *edge* para a IoT também poderá ampliar a vida útil dessa rede IoT, visto que a utilização de computação de borda poderá proporcionar a migração de sobrecargas computacionais e de comunicação de nós com baterias ou fontes de alimentação limitados para outros nós de borda com muito mais recursos [29].

Essas questões, segundo destacado por Yu *et al.* em [29], se tornam mais críticas para aplicações IoT que são *time-sensitive*, ou seja, que necessitam de respostas rápidas em curto espaço de tempo, visto que o resultado será a grande latência nas redes. *Isso ocorre porque os processos de computação precisam ser carregados para a cloud, e a largura de banda limitada e os recursos de rede são ocupados por enormes transmissões de dados, no topo da nuvem já está longe dos usuários finais.*

Como exemplo, cita-se: cuidados de saúde inteligentes, como destacado por Nayala e Kim em [17], visto que exigem informações para a decisão em tempo real; a viabilização de transporte inteligente que necessitam de informação em tempo real conforme defendido por Lin *et al.* em [14]; ou nos casos de *smart grid* que também necessitam da garantia de performance em tempo real, apresentado por Sabella *et al.* em [28]; ou para o funcionamento de *smart city* [16], dentre outros. Nesses casos, o serviço convencional baseado em computação em nuvem não irá satisfazer a demanda impactando a segurança e na resposta a emergências [29].

Destaca-se a importância do desenvolvimento de uma nova arquitetura para IoT em especial no caso das *smart cities*, pois integra as tecnologias da informação e comunicação com todas as infraestruturas físicas de uma cidade, incluído as duas citadas anteriormente (transportes e distribuição de energia – *smart grid*) com o objetivo de proporcionar uma gestão de recursos mais eficiente e, por conseguinte, melhoria do padrão de vida da população. Entretanto, a viabilização das *smart cities* enfrentam vários desafios que impedem o seu rápido desenvolvimento em aplicações de larga escala [16].

> Primeiro, uma delas é a falta de conjunto de dados confiáveis do mundo real para modelagem, análise e técnicas para melhorar as operações que podem ajudar na tomada de decisões para serviços melhorados. Em segundo lugar, processando e analisando dados maciços heterogêneos gerados por diferentes sensores continuam sendo uma questão desafiadora, já que a maioria das plataformas e ferramentas de análise são incapazes de lidar com problemas de compatibilidade e interoperabilidade colocados por esses diversos conjuntos de dados. Em terceiro lugar, projetar plataformas e soluções com recursos que podem coordenar funções de várias dimensões da cidade inteligente sob uma plataforma integrada tem sido o objetivo dos pesquisadores, tanto na academia como indústria [16].

Rose *et al.* e Yu *et al* em [20] e [29], respectivamente, descrevem três diferentes modelos de comunicação para IoT: Comunicação máquina-máquina, Comunicação máquina-*cloud* e Comunicação máquina-*gateway* que serão expostos a seguir.

2.3.1 Comunicação máquina-máquina

Representa a conexão e troca de informações diretamente entre dois ou mais dispositivos, por meio de diferentes redes. A Figura 2.2 mostra um comutador inteligente se comunicando com a luz inteligente sobre o *Bluetooth* 4.0. Esse modelo, em geral, é utilizado em diversas aplicações como sistemas domésticos inteligentes ou controle automático em sistemas elétricos, que se comunicam entre si via envio de pequenos pacotes de dados e requisitos de taxa de dados relativamente baixos. São exemplos: as fechaduras inteligentes, interruptores inteligentes e luzes inteligentes.

Figura 2.2 – Exemplo de comunicação máquina-máquina
Fonte: W. YU *et al.* [29].

.

A comunicação máquina-máquina pode apresentar alguns problemas quando observados sob o ponto de vista dos usuários, como a falta de compatibilidade entre dispositivos que utilizam protocolos diferentes, de diferentes fabricantes, limitando a escolha e a experiência dos usuários.

2.3.2 Comunicação máquina-*cloud*

Rahman *et al.* em [18] define um *Gateway*. Este atua como um *proxy* responsável por conectar dispositivos a redes, incluindo a Internet. Isso é feito com a tradução de diferentes protocolos de comunicação para o TCP/IP, bem como o gerenciamento de registro de serviços e dados mediante análise da informação recebida, permitindo um melhor controle ou até um nível de segurança sobre a rede IoT.

Nesse modelo os dispositivos de IoT demandam os serviços ou armazenam dados um provedor de serviços em *cloud*, essa solução é utilizada devido as limitações de capacidade computacional ou de espaço de armazenamento dos dispositivos, requerendo conexões convencionais com fio ou Wi-Fi, conforme mostrado na Figura 2.3

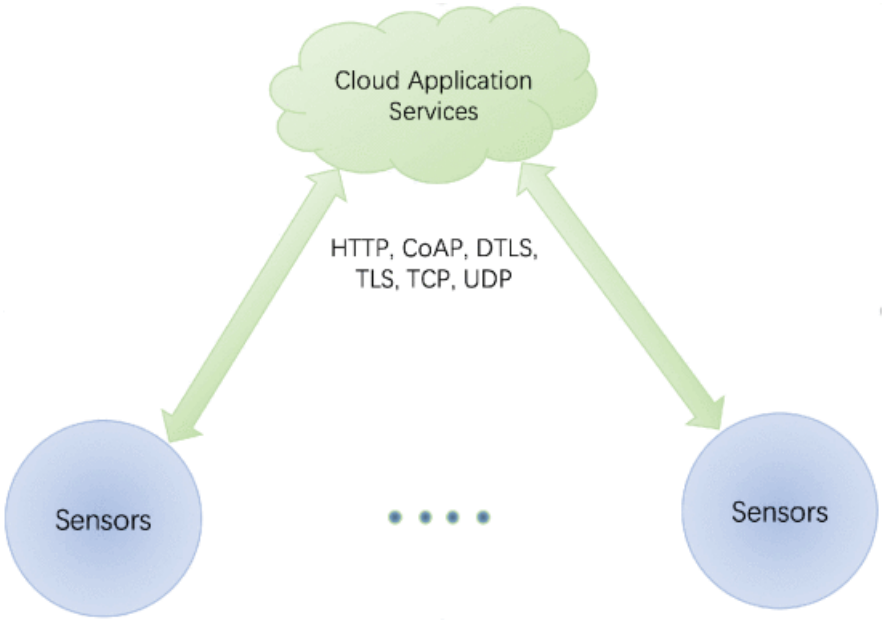

Figura 2.3 – Exemplo de comunicação Máquina-*cloud*
Fonte: W. YU *et al.* [29].

Esse modelo de comunicação soluciona os problemas do modelo Máquina-Máquina, entretanto é dependente da rede tradicional, da largura de banda e dos recursos de rede, o que limitam o seu desempenho, exigindo a otimização da estrutura da rede. Cita-se com exemplo o *Nest Labs Learning Thermostat* e a *Samsung SmartTV*.

2.3.3 Comunicação máquina-gateway

Também conhecido como o modelo *device-to-application-layer gateway* (ALG), é considerado como uma caixa de *proxy* ou *middleware*. A Figura 2.4 apresenta a estrutura desse tipo de comunicação.

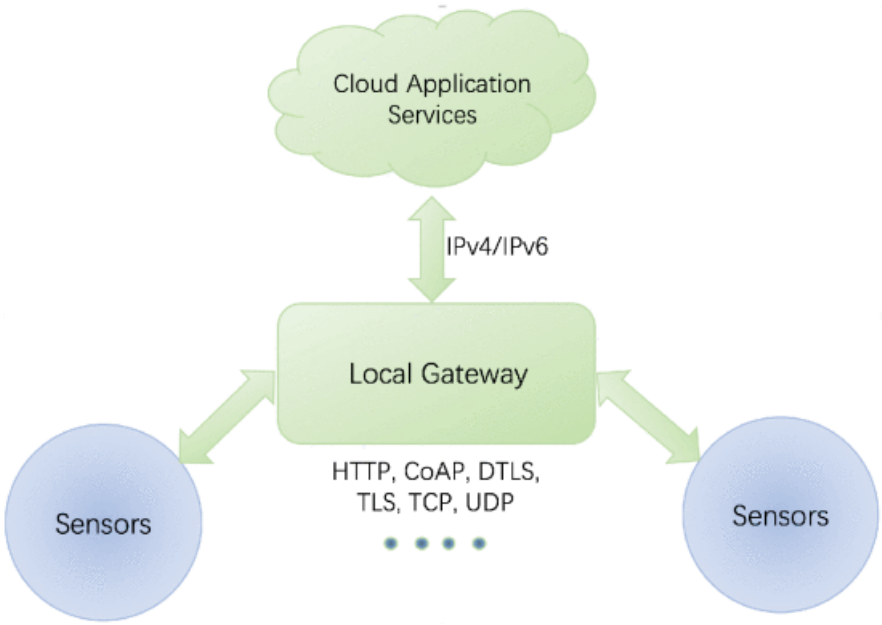

Figura 2.4 – Exemplo de comunicação Máquina-Gateway
Fonte: W. YU *et al.* [29].

Na camada de aplicação, alguns esquemas de verificação de segurança baseados em *software* ou outras funcionalidades, como algoritmos de conversão de dados ou protocolos, são executados em um *gateway* ou outro dispositivo de rede, que atua como uma ponte intermediária entre dispositivos IoT e serviços de aplicação na *cloud*. Isso melhora a segurança e a flexibilidade da rede IoT, migra uma parte da tarefa de computação para a camada de aplicação e reduz significativamente o consumo de energia dos dispositivos IoT. Por exemplo, o telefone móvel inteligente atua como o *gateway*, executando algumas aplicações para se comunicar com os dispositivos IoT e com a *cloud*. Isso aparece no domínio de integridade pessoal, por exemplo, quando os sensores geram dados e se conectam a um *smartphone* pessoal, o dispositivo inteligente criptografa os dados e os carrega nos provedores de serviços em *cloud* [29].

2.4 Edge computing

Em função do rápido aumento do número de dispositivos móveis associada as dificuldades enfrentadas pela *cloud computing* centralizada, a *Edge computing* se apresenta como uma solução chave para esse problema, pois fornece informações de RAN[8] em tempo real, melhorando a QoE[9] dos usuários finais [29].

> A plataforma de computação de borda permite que os nós de borda respondam às demandas de serviço, reduzindo o consumo de largura de banda e a latência da rede. Assim, os operadores de rede podem implementar o RAN na borda para serem manipulados por terceiros, aumentando rapidamente a implantação

[8] RAN - *Radio-Access Network.*
[9] QoE - *Quality-of-Experience.*

> de novas aplicações. Por outro lado, os nós de computação estão operando sob diferentes parceiros de terceiros, dificultando a implementação de esquemas de segurança semelhantes para garantir o mesmo nível de segurança [29].

Segundo Yu *et al.* em [29], existem dois modelos de arquitetura de *edge computing* dominantes: O modelo hierárquico e o modelo definido por software. Para a presente pesquisa será descrito apenas o modelo hierárquico, conforme descrito a seguir.

O modelo hierárquico leva em conta as diferentes distâncias existentes entre os servidores *edge/cloudlet* e os usuários finais, dividindo hierarquicamente a arquitetura de borda, considerando as distâncias e os recursos. Assim, mostra-se adequado para descrever a estrutura de rede da computação de borda.

Yu *et al.* em [29] descreve alguns esforços de pesquisa para a proposição de modelos hierárquicos, entre eles aparecem o de Jararweh *et al.* em [12] e o de Tong *et al.* em [25]. No primeiro, há a proposição de um modelo hierárquico que integra os servidores MEC (Mobile Edge Computing) e as infraestruturas como *cloudlets*, nas quais os usuários móveis podem obter os serviços solicitados sem necessidade da requisição transitar pelo núcleo da Internet, já que o MEC oferece a capacidade de atender às necessidades de computação e armazenamento. No segundo, a proposta é de um modelo de *cloud* de borda hierárquica, que pode ser usado para atender a cargas de pico exigidas de usuários móveis, para tal os servidores de *cloudlets* são implantados na borda da rede e a *cloud* de borda regional é estabelecida como uma hierarquia em árvore. Ao fomentar essas estruturas hierárquicas, é apresentada uma alternativa promissora frente a problemas de latência e de alto tráfego já levantados na seção 1.1.

3 Descrição da Solução

O presente capítulo discorre a respeito da proposta do trabalho, que é um *Middleware* de IoT denominado UIoTBox e tem uma composição modular conforme apresentado na Figura 3.1. O laboratório de Internet da Coisas em que foi desenvolvido esse sistema (UIoT), já utilizava um *Middleware* que continha a capacidade de identificação das entidades presentes na rede e um controle de mensageria, como exposto nos estudos [4, 15, 5]. Inspirado no modo de funcionamento do sistema existente, foi construído o novo *Middleware* capaz de incorporar as funcionalidades já presentes e possibilitar uma escalabilidade para novos objetivos, como, por exemplo, maior detalhamento da caracterização dos dados transitados que permitem melhor análise e processamento, além do maior controle sobre os dados transitados pela rede.

No desenvolvimento do trabalho foi realizada a implementação de módulos que iriam compor o *Middleware* utilizado, uma nova forma de abstração de entidades que compõem a rede IoT e uma biblioteca para uso dessa nova abstração em dispositivos do sistema.

Os módulos do *Middleware* foram especificados e em seguida foram desenvolvidos em *Python* para permitir a implantação hierárquica de *Middlewares* uma rede IoT.

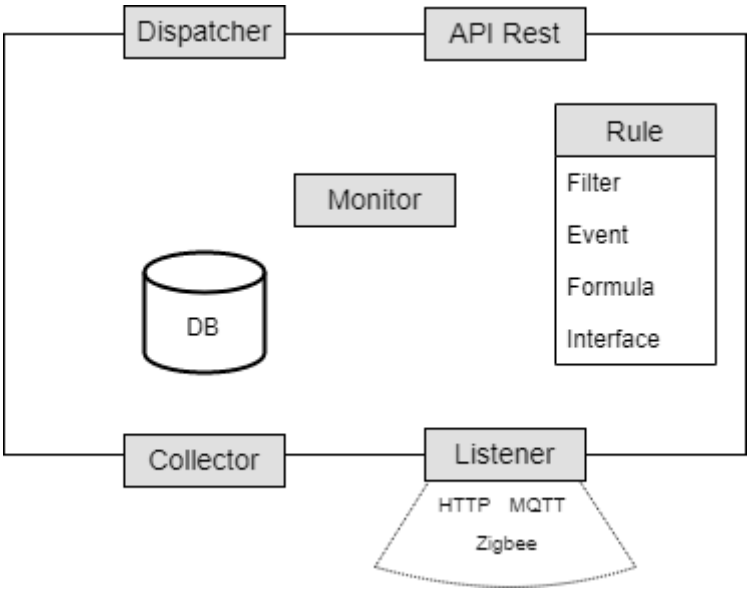

Figura 3.1 – UIoTBox
Fonte: Elaboração dos autores

.

O UIoTBox permite:

- Persistência de dados enviados seguindo a abstração das entidades proposta.
- Persistência de dados de fontes externas passivas, ou seja, que informam dados apenas quando

requisitados.

- Cadastro de fórmulas de sumarização de dados.
- Gerenciamento de eventos para fórmulas de sumarização.
- Envio de resultados de sumarização para fontes externas.

Sua implementação foi feita pensando não apenas para adequação em um sistema hierárquico, mas também de uma forma genérica, permitindo ser utilizado em outras aplicações e arquiteturas IoT.

O UIoTBox é, portanto, a consolidação de alguns módulos em *Python* que, juntos, formam um *Middleware* genérico capaz de suportar regras para análise e envio de dados ou alertas. Ele é capaz de receber, persistir, processar (a partir de sumarizações, compressões) dados de entrada e enviar informações para diversas fontes.

Os dados recebidos por essa entidade podem ser oriundos de dispositivos que conversem a seguindo a abstração das entidades descrita em 3.9. As fontes de dados utilizadas para os casos de teste de foco deste documento se referem a dispositivos ESP-8266, dispositivos simulados em *Python* e outros UIoTBox de hierarquia inferior. A expedição de dados pode ser especificado, possibilitando o envio para qualquer outra entidade que o UIoTBox tenha acesso. Esse envio de dados pode ser feito a partir do cadastro de uma *Rule* (detalhada em 3.6), que possibilita o despacho controlado de dados especificados e ainda possibilitando uma sumarização. Isso permite que entidades de maior nível hierárquico possam obter informações úteis a partir de dados brutos obtidos na borda da rede.

Para execução de sumarização e envio de dados de uma forma customizada, possibilitando aplicação de Edge Computing, o UIoTBox permite um cadastro de regras com um gerenciamento de eventos para acioná-las no momento correto.

Os módulos são apresentados e detalhados nas subseções seguintes.

3.1 Unidade de Persistência

A unidade de persistência é o módulo responsável por guardar os dados necessárias para o funcionamento do UIoTBox na arquitetura proposta. Tais dados estão relacionados à informações dos clientes, dos serviços e dados gerados por tais serviços. Além disso, essa unidade persiste as fórmulas cadastradas no UIoTBox.

Foi utilizado o banco MongoDB para implementação desta unidade. Esta escolha foi feita pela necessidade de utilizar um banco não estruturado e escalável.

Para atender as funcionalidades, projetou-se as *collections* descritas a seguir.

3.1.1 clients

Collection que armazena as informações a respeito dos clientes cadastrados no UIoTBox seguindo a abstração das entidades da arquitetura proposta. Essa *collection* tem como objetivo principal permitir identificar o dispositivo que é fonte de dados para a rede.

O principal identificador do cliente é dado pelo conjunto de **mac** e **chipset**, funcionando como uma espécie de *composite key*[1]

3.1.2 services

Collection que armazena as informações dos serviços cadastrados no UIoTBox além das informações para relacioná-lo com o **cliente** que executa tal serviço.

A partir dos campos do serviço descritos na seção 3.9, é possível identificar o cliente que contém o serviço em questão a partir da consulta da *collection* de **clients** com os parâmetros **mac** e o **chipset** informados no cadastro do serviço. O **number** especificado serve apenas para diferenciar os serviços de um mesmo dispositivo, podendo se repetir em diferentes **clientes**, sendo que a diferenciação é feita pela identificação do próprio cliente pelo **mac** e **chipset**.

3.1.3 datas

Collection que armazena os dados gerados pelos serviços cadastrados no UIoTBox além de informações para relacioná-los, seguindo as abstrações das entidades presentes na arquitetura.

A partir dos campos apresentados na seção 3.9, é possível identificar o cliente que enviou o dado em questão da *collection* **clients** com os parâmetros **mac** e **chipset** informados no cadastro do dado, bem como o **serviço**, desse dispositivo, a partir da junção de **mac** e **chipset** com o **serviceNumber** informado na *collection* **services**

3.1.4 rules

Collection que armazena todas as informações necessárias para o *gateway* executar as regras cadastradas no UIoTBox.

As informações armazenadas em um cadastro de uma regra são:

- *event*

  Contém um objeto que caracteriza o tipo de evento e as condições em que a regra será executada. (Obrigatório)

- *filter*

  Contém um objeto que especifica quais dados devem ser coletados para a checagem de um *event*. (Obrigatório)

[1] Casos em que mais de uma coluna é utilizada para especificar uma chave de uma tabela de banco de dados

- *formula*

  Identifica o tipo de fórmula a ser executada com os dados acumulados a partir da ocorrência de um evento.

- *interface*

  Contém um objeto que caracteriza o tipo de interface de saída para o resultado da regra executada. (Obrigatório)

3.2 *Interface*

Interface é o módulo que permite a conexão a diferentes aplicações externas ao UIoTBox. Essas aplicações e suas formas de conexão podem ser muito diferentes entre si com os mais variados parâmetros e troca de informações, portanto elas são facilmente implementadas e adicionadas ao UIoTBox a partir de uma classe com métodos *get* e *send*. Para cada interface, recebe-se um objeto ***params*** que contém todos os parâmetros utilizados para sua correta execução.

O método *get* é responsável por obter dados de alguma fonte de dados. Este recebe os parâmetros que contém todas as especificações de filtros e modo de comunicação que deverão ser aplicados na obtenção de dados da fonte em questão.

O método *send* refere-se ao método responsável por enviar dados a alguma aplicação externa ao UIoTBox. Esse recebe os parâmetros que contém os dados a serem enviados encapsulados conforme a forma de comunicação.

Para o projeto em questão, implementou-se quatro interfaces: **http_interface; uiot_service; local_mongo; local_dims; listener** e **lempel_zif**.

3.2.1 http_interface

Interface utilizada para enviar e requisitar dados de por HTTP. Os parâmetros que essa interface recebe contém o tipo de requisição de HTTP, sendo ele referente aos métodos GET, PUT e POST. Todos esses métodos também possuem os parâmetros *headers* e *recipient* obrigatoriamente. *headers* são os cabeçalhos e *recipient* se refere ao destino da requisição HTTP a ser executada. Uma requisição GET também necessita do envio de um objeto denominado *url_encoded*, que será utilizado como parâmetro da requisição GET a ser executada, enquanto os métodos POST e PUT necessitam de um objeto denominado *body*, que será utilizado como corpo da requisição a ser executada.

3.2.2 uiot_service

Interface utilizada para se comunicar com outras instâncias de UIoTBoxes, utilizando a abstração das entidades proposta em 3.9, durante a comunicação com os *gateways* IoT presente no no laboratório. Os parâmetros que essa interface recebe se refere aos campos necessários para o preenchimento das informações do serviço gerado pelo evento, sendo eles *serviceNumber*, *serviceParameter* e *serviceUnit*. Esta classe não possui um método *get*.

3.2.3 local_mongo

Interface utilizada para enviar e requisitar dados da Unidade de Persistência do UIoTBox. O método *get* dessa interface recebe como parâmetros um objeto a ser utilizado como filtro para uma consulta a Unidade de Persistência MongoDB presente no UIoTBox. O método *send* dessa interface recebe como parâmetro um objecto que contém a *collection* a ser inserida os dados e os dados em si.

3.2.4 local_dims

Interface utilizada para enviar e requisitar dados da API de comunicação com a Unidade de Persistência do UIoTBox. seu método *send* como parâmetro um objecto que contém a *collection* a ser inserida os dados e os dados.

3.2.5 listener

Interface utilizada para comunicação com o *listener* do UIoTBox. Essa interface só possui o método *get* para a coleta dos dados que estão no *buffer* do *listener* do UIoTBox em questão.

3.2.6 lempel_ziv

Interface utilizada para enviar dados com a utilização de do algoritmo de compactação *Lempel-Ziv*[2]. Ele compacta *array* de dados.

3.3 *Listener*

O *Listener* é o módulo que tem como objetivo receber dados de diferentes dispositivos IoT que conversem com variadas tecnologias. Ele faz a validação da mensagem recebida a partir dos *schemas* de definição de entidade, seguindo a nova abstração das entidades descrita em 3.9. Sua implantação foi fortemente baseada na solução de *gateway* semântico existente no laboratório em que o sistema foi desenvolvido, referenciado em [5]. Esse *gateway* era capaz de compreender mensagens de comunicação através de *Sockets* TCP e UDP, MQTT e ZigBee, e assim fazer todo o gerenciamento de serviços oriundos de dispositivos e conectá-los aos *Middlewares* que os interligam com a Internet. Tal sistema foi evoluído nessa pesquisa para permitir a validação de uma nova forma de abstração das entidades da nova proposta, que é descrito em 3.9, além de fazer o controle de registro sequencial e auditável dessas novas entidades.

O *Listener* é a entidade que faz o controle de cadastro de clientes e serviços no sistema, se atualizando com os já cadastrados na Unidade de Persistência no momento de inicialização e controle dinâmico em seu funcionamento. O registro de dados também é feito por ele, só permitindo registro de dados de clientes e serviços já cadastrados, rastreados pelos campos **mac, chipset** e **serviceNumber**.

[2] Algoritmo baseado na identificação e codificação de valores e padrões em uma estrutura

Os *endpoints* expostos pelo *Listener* se referem a porta de entrada por mensagens HTTP. Ela é dividida em duas raízes: */d*, referindo-se a dados oriundos de dispositivos, e */g*, referindo-se a dados oriundos de outros *Middlewares* da arquitetura. Os campos utilizados nas requisições dessas duas raízes são idênticos, porém existe um campo a mais em todas as requisições feitas em */g* que contém um *token* de autenticação conhecido entre as entidades. Esse *token* é gerado a partir de um registro prévio feito entre *gateways*. A partir dessas raízes, existem *endpoints* para cada entidade descrita na seção 3.9, sendo eles: *../client ../service* e *../data*.

Os dados recebidos em conformidade a nova abstração de entidades são guardados em um *buffer* do próprio módulo. Esse *buffer* é esvaziado a partir da consulta ao *endpoint /getall*, que é atendido pelo módulo, que retorna três listas contendo todos os objetos de clientes, serviços e dados recebidos e guardados no *buffer* desde a última consulta realizada ao *endpoint*.

3.4 *Collector*

O *Collector* é o módulo que permite a coleta de dados de diferentes fontes de interesse e os persiste na unidade de persistência.

As diferentes fontes de dados a serem consultadas para a coleção de dados são especificadas de acordo com a **Interface** utilizada. Portanto, o *Collector* instancia uma interface e utiliza do método *get* para a coleta de dados. No caso do UIoTBox, o *Collector* é utilizado para persistir os dados recebidos pelo *Listener*. Esse módulo fica em constante execução durante o funcionamento padrão do UIoTBox, sempre mantendo a **Unidade de Persistência** atualizada com os dados enviados a ele.

3.5 *Dispatcher*

O *Dipatcher* é o módulo de porta de saída do UIoTBox para diferentes destinos, retornando a quem o instanciou parâmetros do envio efetuado.

Os diferentes destinatários são especificados de acordo com a *Interface* utilizada. Portanto, seu comportamento é antagônico ao do *Collector*, pois o envio efetivo de um dado é feito instanciando-se uma *interface* e utilizando seu método *send* para o envio de dados a um destinatário. Esse módulo é instanciado apenas nos momentos de envio de dados para destinos externos ao UIoTBox.

3.6 *Rule*

Rule é a entidade que define as regras que o UIoTBox deverá seguir sobre os serviços que estão enviando dados a ele. As regras tem como objetivo definir o mecanismo de monitoramento desses serviços e, então, realizar operações e envio dos dados processados. Ela possui 4 objetos, descritos a seguir.

3.6.1 *filter*

O *filter* é o elemento da *Rule* que contém os parâmetros da *query* para filtragem de dados na Unidade de Persistência. Ele é, portanto, o identificador dos serviços de interesse para a regra em questão. Esse filtro pode ser feito ou por um array de nomes de serviços ou por um array com identificadores de serviços, sendo esses um objeto contendo o **mac** e **chipset** do dispositivo que provê o produto juntamente com o **serviceNumber** do serviço de interesse.

3.6.2 *event*

O *event* define o gatilho para aplicação da regra cadastrada. Ele contém um campo *occurrences* que indica o número de ocorrências do evento em questão, sendo que para casos de um número desejado não definido, pode-se informar um valor negativo, permitindo que o número de ocorrências seja infinito. Além desse campo, existem campos que indicam e caracterizam o tipo do evento, descritos a seguir:

- *Time*: Evento que ocorre periodicamente a partir da quantidade de segundos definida no parâmetro *value* de *event*.
- *Quantity*: Evento que ocorre periodicamente a partir da quantidade de amostras presentes no retorno da *query* feita a Unidade de Persistência a partir de *filter* definida do parâmetro *value* de *event*.
- *Bool*: Evento que ocorre quando uma operação lógica sobre os dados retornados da *query* feita na Unidade de Persistência a partir de *filter* resulta em verdadeiro. Essa operação é definida pelos parâmetros:
  - *parameter*: indica quais parâmetros do dado será considerado na operação lógica a ser executada. Trata-se de um *array* de *strings*, em que cada elemento pode ter os seguintes valores: "*values*", "*tag*", "*location*", "*timestamp*", "*clientTime*", conforme detalhado na seção 3.9.
  - *operation*: representa a operação lógica que será realizada. Trata-se de um *array* onde cada elemento é um operador que pode ser: "="(igual), ">"(maior que), "<"(menor que), ">="(maior ou igual que), "<="(menor ou igual que) ou "!="(diferente). O índice do *array* é utilizado com os índices correspondentes dos *arrays* com os valores a serem comparados.
  - *value*, que indica os valores que serão comparados com os dados presentes nos campos especificados em *parameter*. É um *array* onde cada elemento utilizará a operação (*operation*) de mesmo índice com o valor do campo *parameter* também de mesmo índice.
  - O parâmetro *exclusive* é uma *flag* que indica se as operações realizadas devem ser comparadas utilizando *AND* ou *OR*. Quando a *flag* é *True*, todos os testes precisam ser verdadeiros para o evento ocorrer, ou seja, *AND*. Caso contrário, basta que um teste seja verdadeiro para que o evento ocorra, ou seja, *OR*.

3.6.3 *formula*

A *formula* representa a operação que será realizada com os dados obtidos da Unidade de Persistência resultantes da *query* feita a partir do *filter* na ocorrência de um evento, determidado por *event*. Essa operação é identificada como uma *string*. Na implementação atual, o middleware realiza as seguintes fórmulas:

- media: realiza média dos valores das amostras presentes na *query* no momento de ocorrência de um evento.
- max: retorna o maior valor entre as amostras presentes na *query* no momento de ocorrência de um evento.
- min: retorna o menor valor entre as amostras presentes na *query* no momento de ocorrência de um evento.

3.6.4 *interface*

O objeto *interface* contém as informações necessárias para indicar qual *Interface* utilizar (detalhada na seção 3.2. A interface é especificada pelo campo *type* e os parâmetros da interface escolhida é informado em um objeto de nome **params**.

3.7 *Monitor*

O *Monitor* é o módulo responsável por monitorar constantemente a Unidade de Persistência para verificar a ocorrência de eventos e faz execução da *Rule* conforme configuração desta. A Figura 3.2 mostra o fluxo do *Monitor*, onde este consulta as regras cadastradas no UIoTBox, e, para cada regra, verifica os dados na unidade de persistências com os filtros presentes na regra para aferir a ocorrência do evento. Se o evento tiver ocorrido, o *Monitor* executa a fórmula e instancia o *Dispatcher* passando a *Interface* presente na fórmula para enviar o dado processado para seu devido destino. Caso o evento não tenha ocorrido, o *Monitor* continua checando os dados. Como o evento possui uma quantidade de ocorrências, sempre que este ocorrer, o número de ocorrências é decrementado e quando tem valor igual a zero, a regra é excluída.

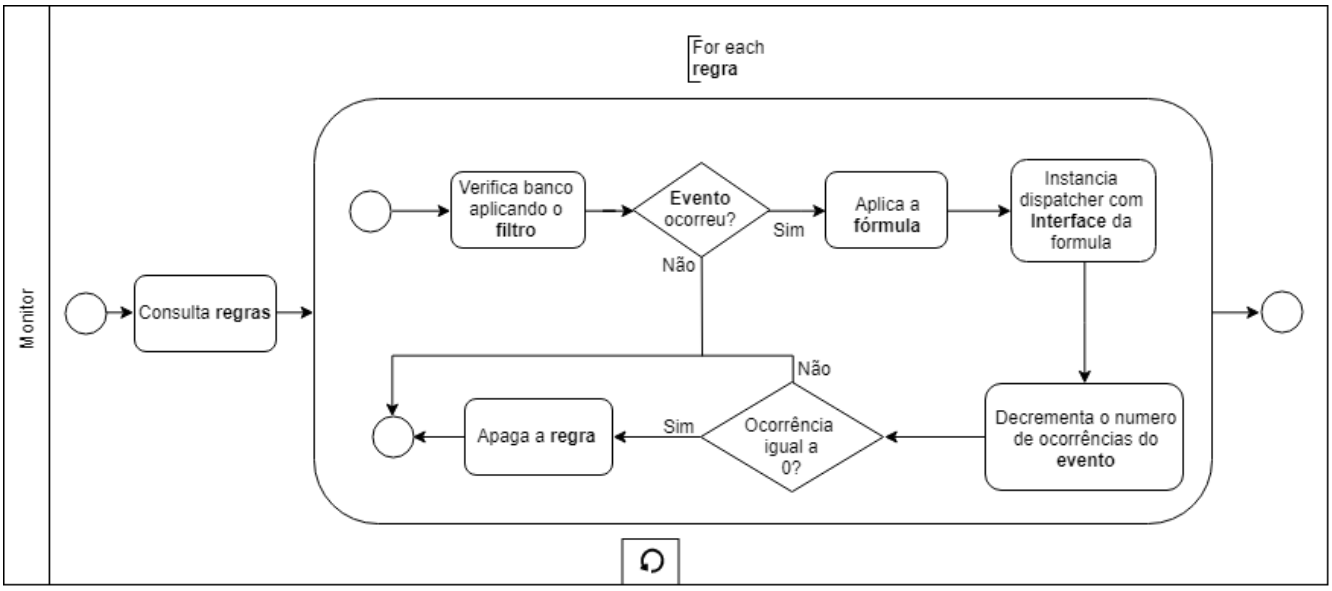

Figura 3.2 – Fluxo Monitor
Fonte: Elaboração dos autores.

Assim, o *Monitor*, é um *script* desenvolvido em *Python* que permanece em constante execução durante o funcionamento do UIoTBox.

3.8 Fluxos de Dados no UIoTBox

Nesta subseção, será ilustrado o funcionamento dos elementos do UIoTBox por meio de exemplos com fluxo de dados em duas situações diferentes.

No primeiro caso (Figura 3.3), é apresentado o fluxo de informações de quando um sensor manda dados ao UIoTBox. Para isso, seguindo todos os passos para a correta relação das entidades na nova abstração, ele se cadastra, cadastra seu(s) serviço(s) e inicia o envio de dados para serem persistidos.

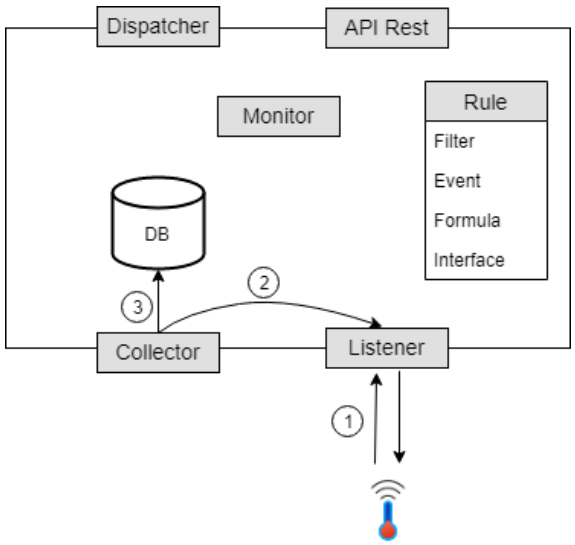

Figura 3.3 – UIoTBox Fluxo 1
Fonte: Elaboração dos autores.

O primeiro passo (número 1 na Figura 3.3), trata-se do cadastramento de um novo cliente no UIoTBox, que, no caso da ilustração, é um sensor de temperatura. Para tal, o dispositivo deve enviar uma mensagem seguindo o fluxo de cadastro para a correta identificação na nova abstração de entidades para o *endpoint /d/client* do *Listener*. A seguir é listado um exemplo do corpo dessa requisição.

```
{
  "mac": "FF:FF:FF:FF:FF:FF",
  "chipset": "AMD 790FX",
  "clientTime": 1000000000.1111,
  "tags": [
    "example-tag"
  ],
  "name": "Raspberry PI",
  "serial": "C210",
  "processor": "Intel I3",
  "channel": "Ethernet",
  "location": "-15.7757876:-48.077829"
}
```

Após o cadastro do dispositivo, este deve cadastrar o(s) serviço(s) que ele provê e que geram dados que serão enviados. O serviço é cadastrado com uma mensagem para o *endpoint /d/service* seguindo o fluxo de cadastro das entidades seguindo a nova proposta abstração. A seguir é listado um exemplo do corpo dessa requisição.

```
{
  "clientTime": 1000000000.1,
```

```
  "tags": [
    "example-tag"
  ],
  "number": 1,
  "chipset": "AMD 790FX",
  "mac": "FF:FF:FF:FF:FF:FF",
  "name": "temperatura",
  "parameter": "temperature",
  "unit": "*C",
  "numeric": 1
}
```

Com o dispositivo e o serviço cadastrado, o sensor pode iniciar a transmissão de dados. Os dados são enviados com uma mensagem seguindo a forma de abstração das entidades para o *endpoint /d/data*. A seguir é listado um exemplo do corpo dessa requisição.

```
{
  "clientTime": 1000000000.1,
  "tags": [
    "example-tag"
  ],
  "sensitive": 1,
  "chipset": "AMD 790FX",
  "mac": "FF:FF:FF:FF:FF:FF",
  "serviceNumber": 1,
  "value": [
    "26"
  ]
}
```

Os dados enviados de um dispositivo para o UIoTBox permanecem em memória no *Listener* até que o *Collector* retire esses dados para então realizar a persistência. Isso é representado pelo passo número 2 da Figura 3.3, em que ocorre a retirada de dados mantidos em *buffer*. Para tal, o *Collector*, utilizando a *Interface listener*, realiza um GET no *endpoint /getall* periodicamente.

O passo 3 da Figura 3.3 ilustra o último passo do recebimento dos dados pelo UIoTBox, a persistência dos registros recebidos. Nesse passo o *Collector* simplesmente passa os dados obtidos no passo anterior para a Unidade de Persistência, a partir do uso da *Interface local_ dims*.

Dessa forma chega-se ao final do fluxo da Figura 3.3, momento em que o dispositivo se cadastra, em seguida cadastra seu(s) serviço(s) e, por fim envia os dados resultantes desses serviços e todas essas informações são persistidas no UIoTBox.

A Figura 3.4 ilustra o fluxo no UIoTBox quando há uma regra cadastrada para enviar um dado sumarizado para o UIoTBox de hierarquia superior após a ocorrência de um evento. Continua-se o

fluxo da situação exposta na Figura 3.3, onde existe um serviço chamado de "temperatura". Para ilustrar este novo fluxo, suponhamos que uma regra foi cadastrada pela API que consiste no envio da média dos serviços de nome igual a "temperatura"a cada 10 amostras recebidas para o UIoTBox superior. Em suma, *Rule* possui *Filter* identificando o nome do serviço igual a "temperatura", um *Quantity Event* com *value* igual a 10 e ocorrência ilimitada, ou seja, igual a -1, *Formula* igual a "Media"e *uiot_service* como *Interface*.

Assim, o passo número 1 ilustrado na Figura 3.4 consiste no *Monitor* checando as regras cadastradas e ativas no sistema. Após checar a regra o Monitor, o passo número 2 ocorre, sendo ele uma consulta a Unidade de Persistência com o filtro da regra e verifica se o evento em questão ocorreu.

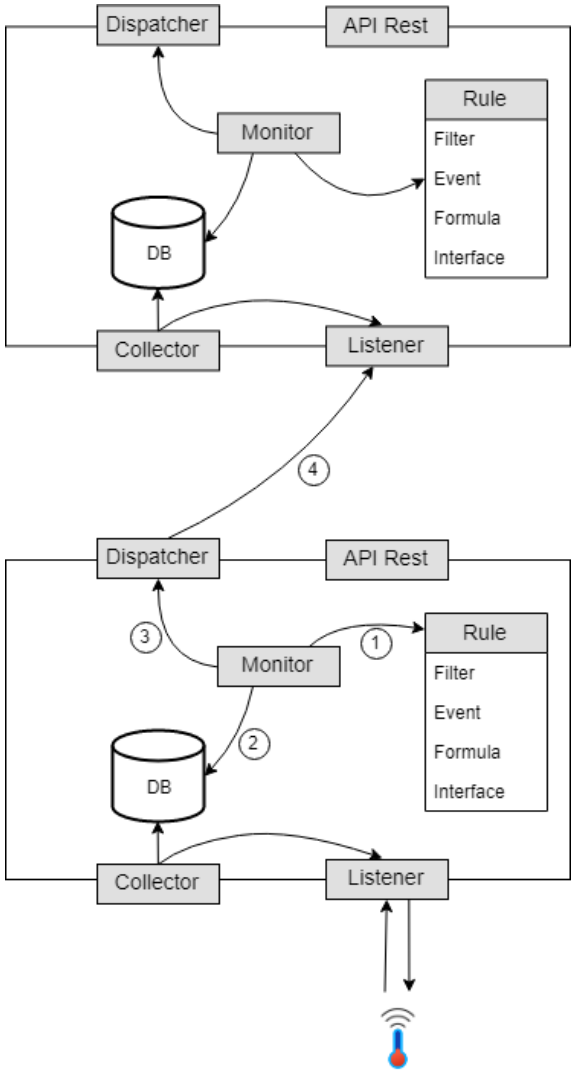

Figura 3.4 – UIoTBox Fluxo 2
Fonte: Elaboração dos autores.

Caso o evento não tenha ocorrido, o fluxo de checagem dessa regra é terminado, mas caso o evento tenha ocorrido (10 novas amostras) o *Monitor* aplica a fórmula (Media) com todo o *dataset*

resultante do evento (10 amostras) e instancia o *Dispatcher* com a *Interface uiot_service* passando como parâmetro o resultado da média de todo o *dataset*, representado pelo passo 3.

Com esta *Interface*, o dado é enviado para o UIoTBox superior como um registro de dado para o endpoint */g/data*, representado pelo passo 4 na Figura 3.4, onde esse dado seguirá o fluxo mostrado na Figura 3.3.

3.9 Abstrações das Entidades da Proposta

O sistema foi desenvolvido seguindo as abstrações de dados do Laboratório UIoT - Laboratório de Internet das Coisas do Departamento de Engenharia Elétrica da Universidade de Brasília. A abstração das entidades da rede descrita evoluiu de trabalhos existentes em formas de descrever e separar as entidades que compõe a rede IoT do laboratório, como exposto no trabalho [23], em que os campos *mac* e *chipset* do dispositivo são utilizados como identificadores de clientes na rede, assim como uma ideia de separação de serviços existentes nesses clientes. Visando escalabilidade para novas aplicações e análises de todos os dados gerados na rede IoT do laboratório, novos campos e separações de entidades foram criados, sendo definidas três entidades que tornam possível o gerenciamento e conexão de dispositivos: Clientes, Serviços e Dados.

A Figura 3.5 mostra o relacionamento entre as entidades supracitadas, em que um *Client* possui um ou mais *Service*(s) que produzem os dados em si, representados por *Data*.

Figura 3.5 – *Client*, *Service* e *Data*
Fonte: Elaboração dos autores

.

Os Clientes são os dispositivos físicos presentes no sistema. Eles são responsáveis por coletar e produzir dados referentes ao seu ambiente e enviá-los a um *gateway* IoT. Um cliente é composto por:

- **mac**

  Indica o endereço MAC do dispositivo. (Obrigatório)

- **chipset**

  Indica o *chipset* do dispositivo. (Obrigatório)

- **clientTime**

  Indica o *timestamp* informado pelo usuário no momento de envio dos dados de cadastro de um **cliente**.

- **name**

  Indica um nome atribuído pelo próprio dispositivo.

- **location**

  Indica a localização do dispositivo.

- **channel**

  Indica o canal que foi enviado os dados enviados pelo dispositivo. (Obrigatório)

- **processor**

  Indica o tipo de processador utilizado pelo dispositivo.

- **serial**

  Indica o número serial do processador utilizado pelo dispositivo.

- **tags**

  É um *array* de *strings* para etiquetar o dispositivo cadastrado de forma livre.

Os Serviços são a abstração de capacidades de dispositivos. Eles são responsáveis por identificar características dos serviços que são providos pelos clientes. Um serviço é composto por:

- **mac**

  Indica o endereço MAC do dispositivo. (Obrigatório)

- **chipset**

  Indica o *chipset* do dispositivo. (Obrigatório)

- **clientTime**

  Indica o *timestamp* informado pelo usuário no momento de envio dos dados de cadastro do **serviço**.

- **name**

  Indica o nome do serviço atribuído pelo próprio **cliente**. (Obrigatório)

- **number**

  Indica um identificador único do serviço do cliente. (Obrigatório)

- **numeric**

  Indica se os dados enviados pelo serviço são numéricos ou não. (Obrigatório)

- **unit**

  Indica a unidade física dos dados gerados e enviados pelo serviço.

- **tags**

  É um *array* de *strings* para etiquetar o dispositivo cadastrado de forma livre.

Os Dados fazem referência ao produto resultante dos Serviços dos Clientes. Eles registram as informações do ambiente ou do processamento de dados coletados por dispositivos em capacidades especificadas. Um Dado é composto por:

- **mac**

  Indica o endereço MAC do dispositivo. (Obrigatório)

- **chipset**

  Indica o *chipset* do dispositivo. (Obrigatório)

- **clientTime**

  Indica o *timestamp* informado pelo usuário no momento de envio dos dados de cadastro do **serviço**.

- **sensitive**

  Indica se o dado é sensível, ou seja, deve ter prioridade para envio ou processamento.

- **serviceNumber**

  Indica o número identificador único do serviço no dispositivo que gerou o dado enviado.

- **value**

  É um *array* que contém o(s) dado(s) referente ao serviço

- **time**

  Indica o *timestamp* do momento de envio do dado.

- **tags**

  É um *array* que contém *strings* para etiquetar o dado registrado de forma livre.

É possível identificar a qual *Client* um *Service* a partir dos campos *mac* e *chipset*. Já para *Data*, é possível identificar a qual *Service* que o produziu a partir do identificador *serviceNumber* junto com os campos *mac* e *chipset*, sendo também possível a idenficação do dispositivo que enviou o dado.

3.10 Implementação de uma Biblioteca para *Devices*

Uma biblioteca foi implementada em C-Arduino e foi integrada a Arduinos e dispositivos ESP-8266 que utilizassem o *gateway* do laboratório UIOT como destinatário para o envio de dados. Seu objetivo era gerenciar toda a parte de comunicação com o *gateway*, realizando o cadastro de cliente, serviço e envio de dados.

As bibliotecas foram implementadas com o uso de hierarquia de classes e uso de métodos abstratos. Com isso, foi possível implementar classes com métodos específicos para o envio dos dados em si, porém utilizar os métodos de construção das requisições que seguem as comunicações que controem entidades descritas na seção 3.9. A partir disso, vários protocolos de comunicação puderam ser implementados e adicionados a biblioteca.

Na biblioteca em questão, foi desenvolvida a capacidade de envio de dados por HTTP[3], MQTT[4] e ZigBee por meio do uso de uma classe abstrata que contém os métodos *register_client*, *register_service* e *register_data*. Esses métodos são sobrescritos por outras classes que possibilitam a comunicação por diferentes protocolos/modos de comunicação.

As classes desenvolvidas para atender dispositivos **Arduinos** e **ESP**, possibilitam o desenvolvimento de clientes com serviços diversos seguindo todo o fluxo de cadastro de clientes, serviços e dados, tanto com a construção das requisições quanto com o efetivo envio de dados de uma forma fácil. Para isso, é necessário apenas os seguintes passos:

1. Criação de um cliente.
2. Criação de serviços.
3. Envio de dados.

O primeiro passo é feito instanciando uma classe que contém o protocolo de comunicação desejado, seguido de uma inicialização do objeto pelo método *init*, que recebe como parâmetro o nome do cliente criado. Essa inicialização cria todos os parâmetros obrigatórios informados descritos na seção 3.1.1.

O segundo passo é feito a partir do método *create_service* da instância do cliente criado na etapa um. Esse método recebe como parâmetro todos os dados obrigatórios indicados na 3.1.2. O usuário da biblioteca precisa, portanto, apenas informar esses parâmetros na chamada do método.

O terceiro passo permite o envio de dados para o *Middleware*. Nesse último passo é feito todo o gerenciamento do fluxo de mensagem de registros de clientes e serviços entre o dispositivo e o *Middleware*. Esse gerenciamento é feito a partir da análise das respostas (caso existam) do *Middleware*, verificando a necessidade do envio ou reenvio das requisições de cadastro de cliente e serviço para o correto envio da requisição com os dados gerados pelos serviços internos.

A biblioteca implementada para dispositivos ESP também possui a implementação de bibliotecas externas para melhor administração dos dispositivos criados. Dentre elas, temos o *WiFi Manager*, que permite a escolha e troca de rede *WiFi* utilizada pelo dispositivo e o *OTA*, que permite a atualização do código utilizado pelo próprio dispositivo a partir da rede *wireless* utilizada. Todas essas funcionalidades em conjunto com a lógica de comunicação e gerenciamento de informações desenvolvida ocupou o total de 9% do espaço reservado para programas.

[3] Hypertext Transfer Protocol
[4] Message Queuing Telemetry Transport

3.11 Estação Meteorológica

Os cenários de testes descritos na seção 4.2 referenciam serviços que enviam dados de temperatura, umidade e luminosidade. Para a captura dos dados de temperatura e umidade foram utilizados sensores DHT11. A leitura desses dados e envio para o *Middleware* foram realizados com dispositivos ESP8266, sendo que, para tal atividade, utilizou-se a biblioteca citada na seção 3.10.

Para conectar o sensor ao ESP8266, projetou-se um circuito de forma compacta com o objetivo de ocupar o menor espaço possível. O esquemático desse circuito está representado na Figura 3.6. Foi utilizado uma placa perfurada de fenolite. Nos pinos representados com a cor vinho, deve ser inserido o ESP8266, conforme orientação do VCC[5] e GND[6]. O sensor DHT11 deve ser inserido nos pinos azuis também seguindo as orientações de VCC e GND da placa. As conexões presentes no circuito, apenas ligam as alimentações no sensor e no ESP além de associar os sinais de saída e entrada dos respectivos. A alimentação do circuito é feita com baterias de lítio. Com o objetivo de otimizar o uso da fonte de energia, foi desenvolvido um modo de hibernar o ESP8266. Para tal, foi necessário soldar o pino de *reset* da CPU do dispositivo com o pino de *GPIO* 2, responsável por mandar um sinal no momento de habilitação, permitindo que o ESP seja acordado e possa novamente enviar dados.

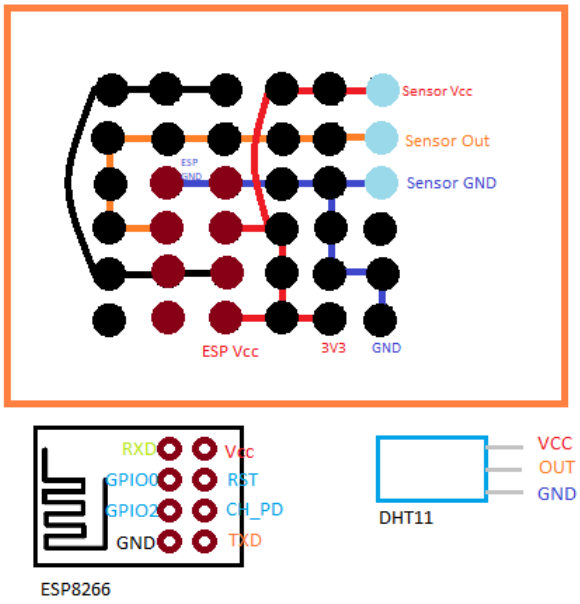

Figura 3.6 – Esquemático Circuito Estação Meteorológica
Fonte: Elaboração dos autores.

A materialização deste esquemático está representada na Figura 3.7. Nela, temos tanto o

[5] VCC - *Common Collector Voltage*.
[6] GND - *Ground*.

circuito projetado quanto os dispositivos utilizados e corretamente posicionados.

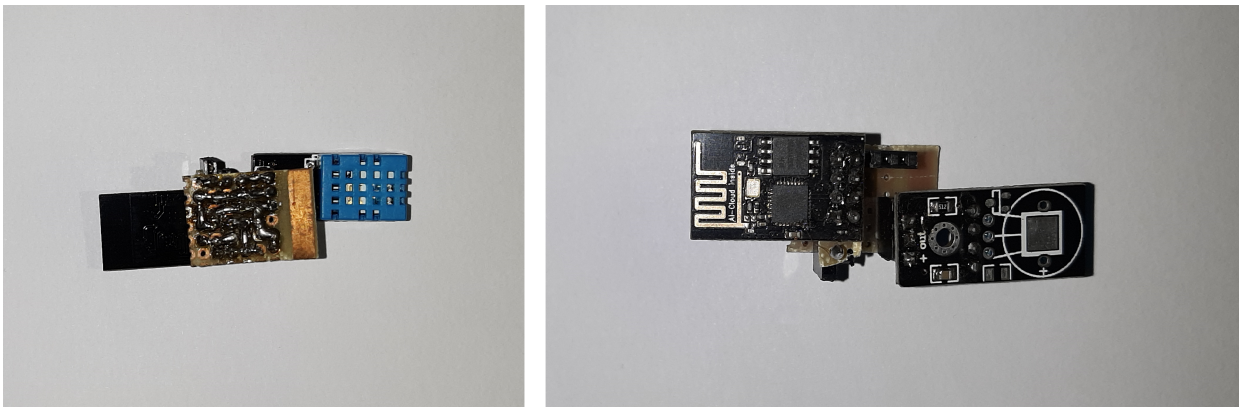

Figura 3.7 – Estação Meteorológica Hardware
Fonte: Elaboração dos autores.

Desta forma, foi possível a construção de uma estação meteorológica portátil de baixo consumo de energia com capacidade de medir e transmitir dados de temperatura e umidade seguindo o fluxo e o esquema de mensagem que abstraem as entidades de comunicação do *Middleware* (Figura 3.8).

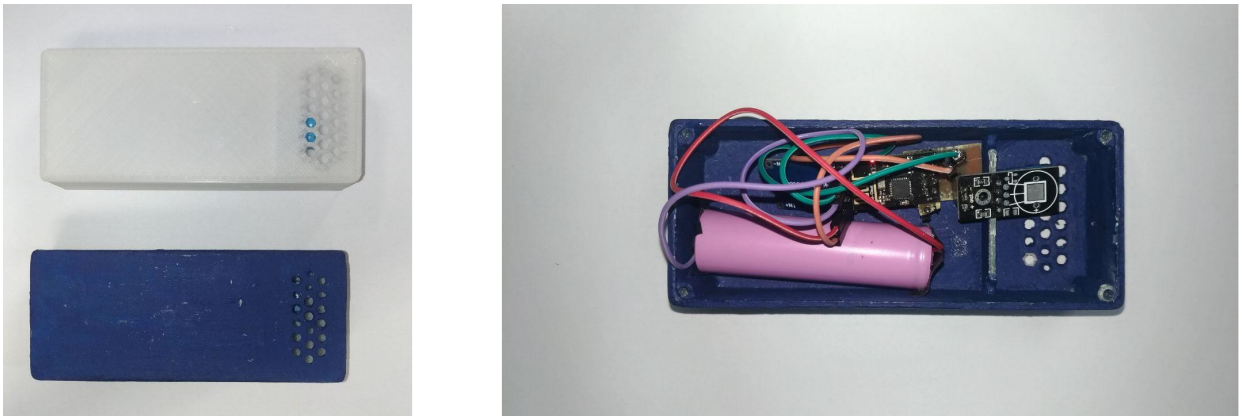

Figura 3.8 – Estação Meteorológica
Fonte: Elaboração dos autores.

4 Resultados e Análise

Neste capítulo, são apresentados os instrumentos utilizados para todo o desenvolvimento da solução, os cenários de teste para a validação da arquitetura proposta e a execução desses cenários com a exposição e análise dos resultados obtidos.

4.1 Instrumentos Utilizados

Para esta pesquisa, utilizou-se as seguintes ferramentas:

- Python

  Linguagem orientada a objetos utilizada para desenvolvimento dos módulos utilizados pelo *Middlewares* hierarquizado, sensores simulados e APIs[1] para obtenção e cadastro de dados persistidos em bancos de dados.

- ESP-8266

  Microcontrolador que contém CPU de 80Mhz com 14 pinos de I/O[2] digitais, sendo apenas dois mapeados para pinos de uso. Utilizado para montagem de dispositivos capazes de coletar dados e enviá-los para um *Middleware* IoT.

- MongoDB

  Banco de dados *open source* não estruturado orientado a documentos que são guardados em *collections*[3], utilizado para persistir os dados para funcionamento do sistema e dados gerados por dispositivos.

- Docker

  Tecnologia de *software* de abstração e automação de virtualização de nível de sistema operacional que permite a separação de aplicações em contêineres. Foi utilizado para a implantação dos diferentes módulos do sistema implementado.

- Prometheus

  Solução *open source* de coleta de métricas para monitoramento de sistemas. Utilizado para a coleta de métricas para comparação entre as arquiteturas da pesquisa.

- Grafana

  Sistema de monitoramento e análise de métricas dos sistemas utilizados.

- *Digital Humidity Tempeture Sensor* (DHT11).

  Utilizado para obtenção de valores de umidade e temperatura do ambiente.

[1] API - *Application Programming Interface.*
[2] I/O - *Input/Output.*
[3] *Collections* são análogas a tabelas em banco de dados relacionais.

- *Light Dependent Resistor*

  Utilizado para obtenção de indicadores de iluminação do ambiente.

- C-Arduino

  Linguagem de programação para dispositivos Arduino e ESP. Utilizado para criação de bibliotecas de comunicações com *Middlewares* IoT, bem como dispositivos geradores de dados para a rede implementada.

- Raspberry Pi3

  Hardware portátil com CPU Broadcom BCM2837, 1GB de RAM[4] e 30 pinos GPIO[5] com sistema operacional Rasbian. Utilizado para implantação de clientes e *Middlewares* da arquitetura proposta.

4.2 Cenários de Teste

A comparação entre a arquitetura de rede em *cloud* e a arquitetura hierárquica geograficamente distribuída proposta foi feita a partir de cenários de testes. Esses cenários abordam iterações com aplicações em cenários reais no contexto de uma cidade. Os cenários desenvolvidos foram:

1. Cenário 1

   Uma rua deseja obter a média de umidade dos últimos 30 minutos de todas as casas presentes nela.

2. Cenário 2

   Um bairro deseja identificar momentos em que a média de temperatura nas casas de uma rua estiver maior que 25°C.

3. Cenário 3

   Um bairro deseja receber o valor máximo de um serviço de luminosidade a cada 1.000 amostras coletadas, sendo que o serviço de luminosidade manda dados em uma frequência não periódica.

A Figura 4.1 representa a implementação de uma arquitetura hierárquica com utilização do UIoTBox.

[4] RAM - *Random Access Memory.*
[5] GPIO - *General Purpouse Input/Output.*

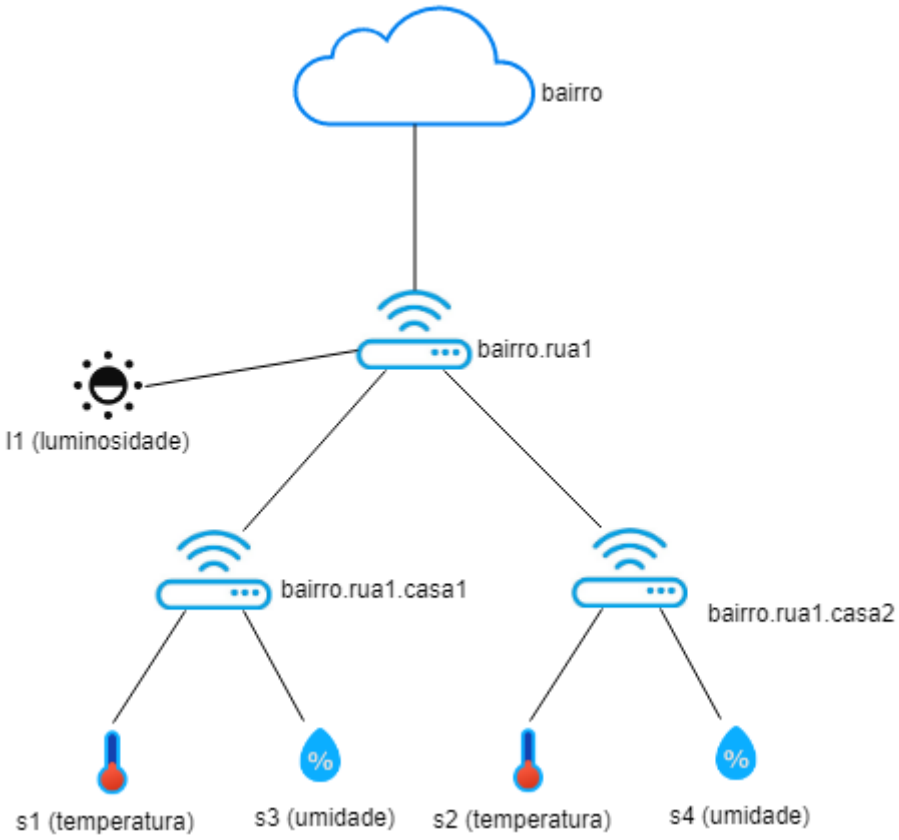

Figura 4.1 – Arquitetura Hierárquica
Fonte: Elaboração dos autores

.

É possível verificar que na Figura 4.1, exitem quatro *cloudlets* com implantações de UIoTBoxes. Estas foram pensadas para aplicações no contexto de uma *Smart City* e assim aparece:

- Um Bairro, representado pelo identificador bairro.

  O bairro representa a entidade de maior nível hierárquico na rede em questão. Para os testes executados, implantou-se um UIoTBox em uma VPS[6] com 4 núcleos de CPU e 4GB de memória RAM.

- Uma Rua, representada pelo identificador bairro.rua1.

  Essa rua representa outra entidade de nível inferior apenas ao bairro. Nela, existe um serviço de luminosidade e duas entidades de níveis inferiores, representados como bairro.rua1.casa1 e bairro.rua1.casa2, que serão futuramente descritas. Para os testes executados, implantou-se um UIoTBox em um *Raspberry Pi*.

- Duas Casas representadas pelo identificadores bairro.rua1.casa1 e bairro.rua1.casa2.

  Essas entidades representam as entidades de menor nível hierárquico. Ambas simulam casas que estão presentes em uma rua de um bairro, que está representado pelo bairro.rua1. Em

[6] VPS - Virtual Private Server

cada uma delas, exitem dois serviços: um de temperatura e um de umidade relativa do ar. Para os tetes executados, implantou-se um UIoTBox em um *Raspberry Pi*.

A Figura 4.2 representa a implementação de uma rede em arquitetura *cloud* com as mesmas entidades e serviços já descritas. A imagem separa as entidades casas e ruas, entretanto essa divisão é apenas simbólica, pois no funcionamento da rede em questão, não há divisão clara dessas entidades. A *cloud* em si representa o bairro na rede, e todos os dados serão enviados a ela, bem como consultados para obtenção a obtenção de dados.

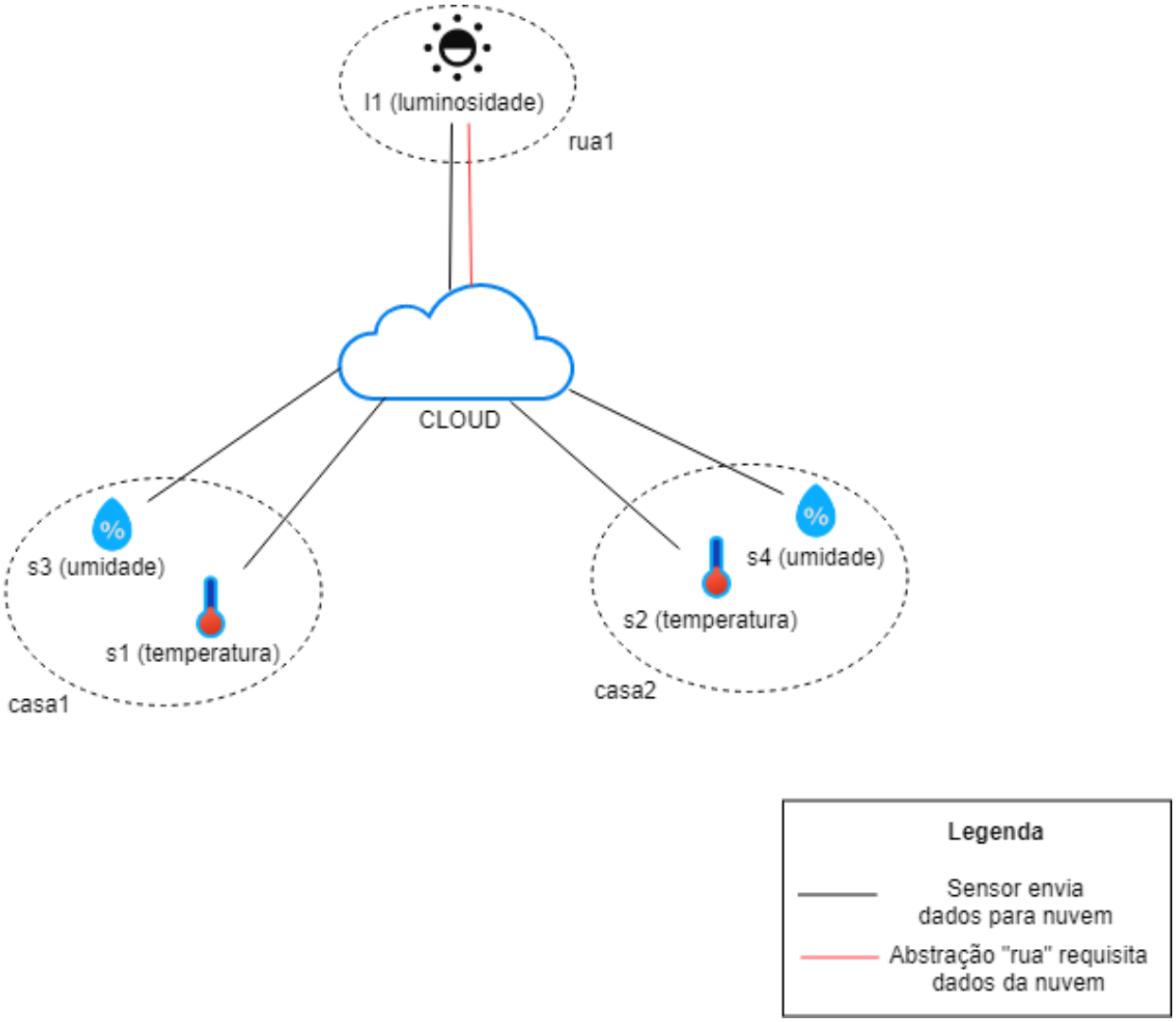

Figura 4.2 – Arquitetura *cloud*
Fonte: Elaboração dos autores

.

Implantou-se um UIoTBox em um *Raspberry Pi* e em um servidor de maior capacidade de processamento para representar a *cloud* dessa arquitetura. Os mesmos serviços e dispositivos utilizados para as casas e as ruas da arquitetura hierárquica já descrita direcionaram seus dados para essa *cloud*. As entidades Rua(s) e Casa(s) são apenas abstrações, pois não são entidades

físicas da arquitetura. A execução dos casos de testes que necessitam dessas entidades foram feitos a partir de códigos em *Python* que executavam requisições para obtenção dos dados da *cloud* e processamento.

As variáveis utilizadas nesta pesquisa foram: temperatura, umidade e níveis de luminosidade. Os valores dessas variáveis foram capturados por dispositivos capazes de obter esses dados. No cenário da rede hierárquica, utilizou-se dois sensores DHT11 conectados a um ESP8266 para medir temperatura e umidade do ambiente e enviá-los para UIoTBoxes de casas. Utilizou-se também um sensor de luminosidade conectado ao ESP8266 para enviar as leituras ao UIoTBox de rua. Os mesmos dispositivos foram utilizados no cenário *cloud*, mas dessa vez todos mandam dados para o mesmo UIoTBox.

4.3 Produção dos Resultados

Os cenários de testes presentes na seção 4.2 foram executados para comparação das duas arquiteturas. Para isso, utilizou-se *Raspberries Pi* para atuarem como as entidades de casas e rua, conforme mostrado na Figura 4.1. Escolheu-se utilizar esses dispositivos para não preocupação com uma camada de virtualização na coleta das métricas de comparação, além de estarem disponíveis para uso no ambiente de desenvolvimento. A adaptação do sistema para arquitetura de 32bit em ARM[7] também se mostrou importante, tendo em vista que tal *hardware* é muito utilizado no laboratório de pesquisa, além de possibilitar eventuais utilizações de tecnologia como *bluetooth*. Os dispositivos utilizados para alimentar os sistemas com dados coletados sobre o ambiente foram ESP-8266, conforme descrito em 3.11, a partir do uso da biblioteca desenvolvida.

Como a pesquisa assumia que todas as entidades estavam conectadas em rede, todos os dispositivos, tanto *Raspberry Pi* quanto ESP-8266, se conectaram a mesma rede Wifi, exceto a entidade de bairro, que foi implantada em uma VPS externa a rede da Universidade de Brasília. A conexão para esse último foi possível com a configuração de um *proxy* reverso, visto que a rede disponibilizada pela universidade bloqueava qualquer conexão a portas não padrões, amplamente utilizadas por quase todos os sistemas da solução proposta nessa pesquisa.

Utilizou-se contêineres de *Docker* para as implantações dos serviços necessários para os testes executados. Os contêineres que em conjunto formam o UIoTBox são: *engine*, que contém os módulos *API Rest, Collector, Dispatcher, Monitor* e Unidade de Persistência; *dims*, que se refere a API que gerencia todos os dados de clientes serviços e dados na Unidade de Persistência; e *gateway*, que se refere ao módulo *Listener*. Primeiramente, foi observado o comportamento do uso de CPU e do tráfego da rede e depois um estudo quantitativo do envio de dados dos serviços e dos cenários de texto.

A implantação do caso de teste de hierarquia necessitou do cadastro de regras que possibilitariam a execução dos cenários de teste. Nas entidades Casa 1 e Casa 2, as mesmas regras foram cadastradas:

[7] ARM - Advanced RISC Machine, que se refere a uma família de arquiteturas que visam a simplificação, flexibilidade e customização em sistemas embarcados

1. Cálculo da média de Umidade a cada 30 minutos. O cadastro dessa regra feito pela API de regras está apresentado a seguir.

```
{
    "filter": {
        "service_names": [
            "Umidade"
        ]
    },
    "event": {
        "type": "Time",
        "value": 1800,
        "occurrences": 10
    },
    "interface": {
        "type": "UIoTService",
        "params": {
                        "serviceNumber": 90,
                        "serviceName": "MediaUmidade",
                        "serviceParameter": "Humidity",
                        "serviceUnit": "%"
                }
    },
    "formula": "Media"
}
```

2. Cálculo da média de Temperatura a cada 10 minutos. O cadastro dessa regra foi feito pela API de regras está apresentado a seguir.

```
{
    "filter": {
        "service_names": [
            "S1"
        ]
    },
    "event": {
        "type": "Time",
        "value": 600,
        "occurrences": 50
    },
    "interface": {
        "type": "UIoTService",
        "params": {
                        "serviceNumber": 91,
                        "serviceName": "MediaTemperatura",
                        "serviceParameter": "Tempeture",
                        "serviceUnit": "*C"
                }
    },
    "formula": "Media"
}
```

A entidade Rua 1 obteve o cadastro das seguintes duas regras:

1. Informar médias de temperaturas maiores que 25 °C. O cadastro dessa regra foi feito pela API de regras está apresentado a seguir.

```
        {
            "filter": {
                "service_names": [
                    "MediaTemperatura"
                ]
            },
            "event": {
                "type": "Bool",
                "value": [
                    25
                ],
                "parameter": [
                    "value"
                ],
                "operation": [
                    ">"
                ],
                "occurrences": -1,
                "exclusive": true
            },
            "interface": {
                "type": "UIoTService",
                "params": {
                                "serviceNumber": {{serviceNumber}},
                                "serviceName": "TemperaturaAlta",
                                "serviceParameter": "Tempeture",
                                "serviceUnit": "*C"
                        }
            },
            "formula": "Max"
        }
```

2. Informar o valor máximo do serviço de luminosidade a cada 1000 amostras. O cadastro dessa regra foi feito pela API de regras está apresentado a seguir.

```
        {
            "filter": {
                "service_names": [
                    "Luminosidade"
                ]
            },
            "event": {
                "type": "Quantity",
                "value": 1000,
                "occurrences": -1
            },
```

```
12              "interface": {
13                  "type": "UIoTService",
14                  "params": {
15                                  "serviceNumber": {{serviceNumber}},
16                                  "serviceName": "MinLum",
17                                  "serviceParameter": "Luminosity",
18                                  "serviceUnit": "Lux"
19                          }
20              },
21              "formula": "Max"
22          }
```

O caso de teste da arquitetura *cloud* centralizada, representada na Figura 4.2, utilizou uma instância do UIoTBox no mesmo Raspberry Pi que correspondeu a entidade bairro.rua1 da arquitetura hierarquizada. Para a realização dos cenários, foi desenvolvido um cliente em *Python* que solicitava os dados com as mesmas *rules* dos cenários do caso anterior. O cliente foi implantado em um outro *Raspberry Pi 3*.

A partir dessa arquitetura, coletou-se informações sobre o uso de CPU nos *hardwares* utilizados como *Middlewares* de cada arquitetura. O intuito dessa coleta foi analisar como seria o comportamento do uso dessa métrica nas duas arquiteturas para a execução da mesma atividade, explorando benefícios de processamento distribuído, possibilitando a não necessidade de recursos computacionais muito grandes no núcleo da internet para atender a clientes na borda ou para obtenção de dados transformados. Coletou-se, também, métricas referentes ao tráfego de rede, obtendo gráficos comportamentais de uso de banda para envio de dados como os dados transitados em si, permitindo também uma comparação quantitativa desses dados.

4.4 Uso de CPU

4.4.1 Caso Arquitetura de *Cloudlets* Hierárquicas

Utilizando a arquitetura hierárquica representada em 4.1, executou-se todos os cenários descritos em 4.2 simultaneamente e extraiu-se informações de tráfego de rede e de consumo de CPU. Os gráficos presentes nas imagens desta seção estão separado por contêineres do *Docker*, são eles: dims (verde), *engine* (amarelo) e *gateway* (azul).

A Figura 4.3 mostra o comportamento do uso de CPU pelo *Middleware* que representa a Casa 1 durante a execução dos cenários de teste. O *Hardware* utilizado foi um Raspberry Pi3.

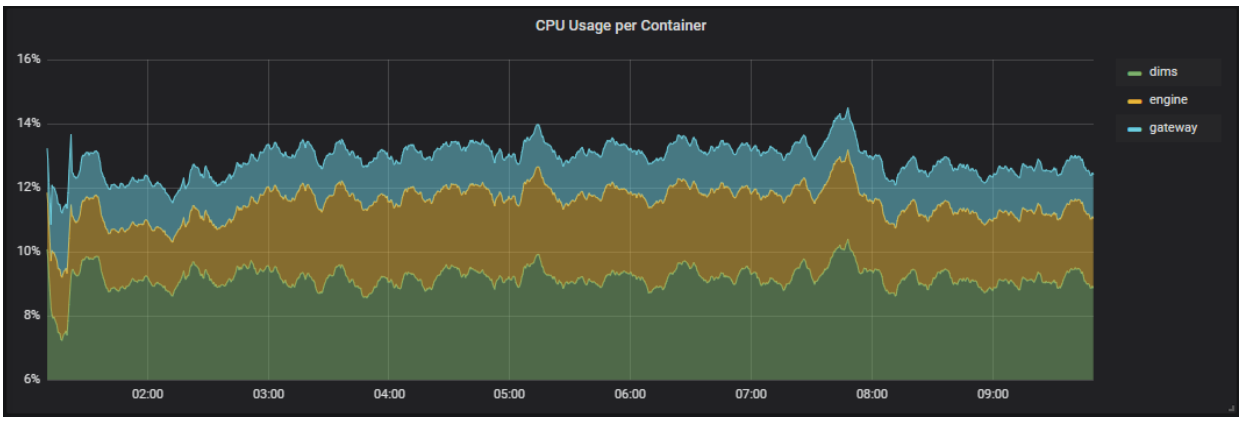

Figura 4.3 – CPU Casa 1 - Arquitetura Hierárquica
Elaboração dos autores.

Nota-se que durante todo o período de execução, a média do uso do *dims* não ultrapassou 10%, com alguns picos que não ultrapassam 12%. A média do uso do *engine* ficou abaixo dos 12% com picos que não ultrapassam 14%. Já no caso do *gateway*, o uso médio ficou abaixo dos 14% com picos que não passaram de 16%.

A Figura 4.4 mostra o comportamento do uso de CPU pelo *Middleware* que representa a Rua1 durante a execução dos cenários de teste. O *Hardware* utilizado também foi um Raspberry Pi3.

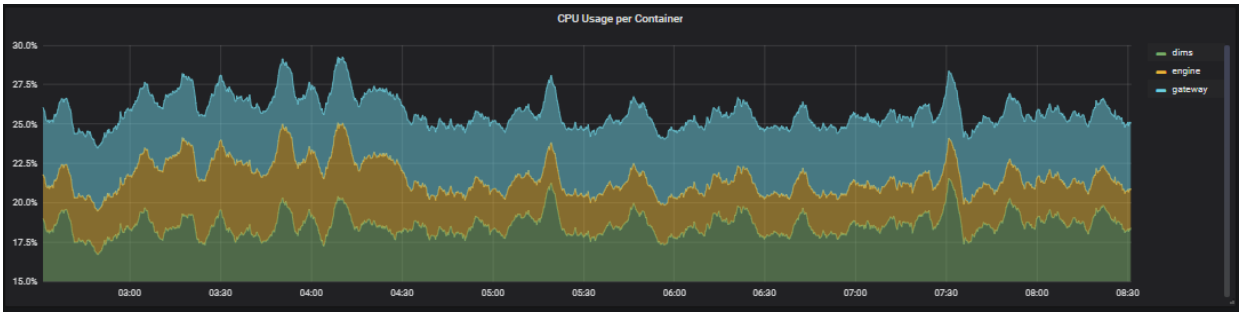

Figura 4.4 – CPU Rua 1 -Arquitetura Hierárquica
Elaboração dos autores.

Observa-se que os picos do uso de CPU pelo dims não ultrapassaram $22,5\%$. Os picos de uso pelo *engine* ficaram abaixo dos 25%.No caso do *gateway*, os picos não passaram de 30%. Esse *Hardware* obteve um maior consumo de recursos de processamento, pois era o responsável pela execução de um número maior de regras e era o que mais recebia informações, visto que o dispositivo de luminosidade enviava a maior quantidade de dados.

A Figura 4.5 mostra o comportamento do uso de CPU pelo *Middlware* que representa o bairro durante a execução dos cenários de teste. Como este *Middleware* foi implantado em um servidor hospedado na *cloud* em uma máquina com quatro núcleos de CPU, a porcentagem do uso se mostra muito reduzida se comparada a de um Raspberry Pi.

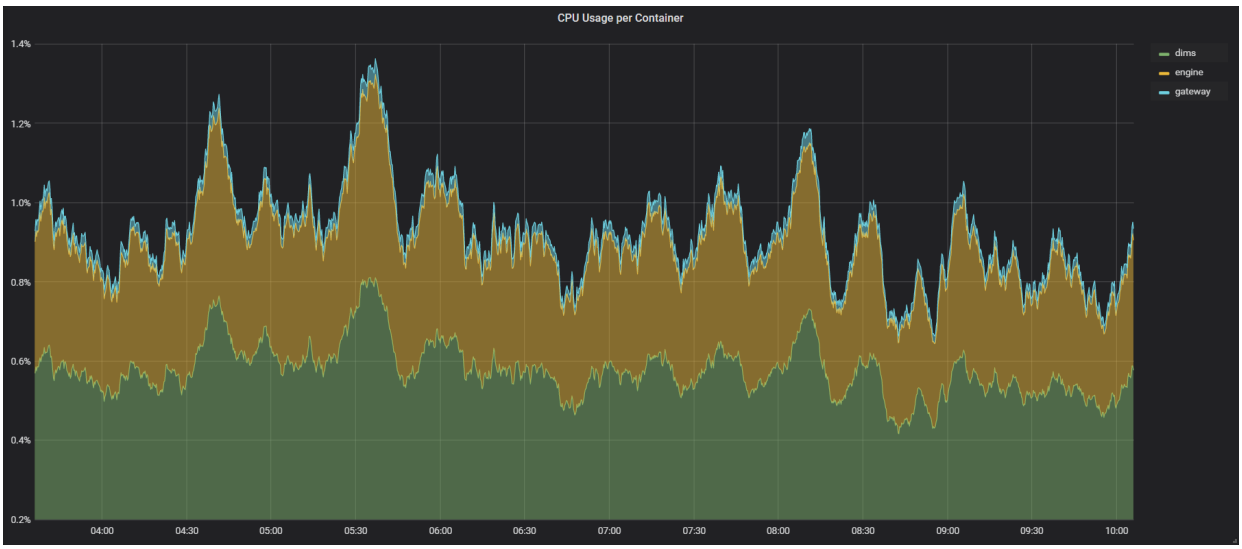

Figura 4.5 – CPU Bairro - Arquitetura Hierárquica
Elaboração dos autores.

Assim, percebe-se que os picos do uso de CPU pelo *dims* não ultrapassaram 1% e os picos de uso pelo *engine* e *gateway* ficaram abaixo dos 1,4%.

4.4.2 Caso Arquitetura *Cloud*

No caso da Arquitetura Hierárquica, a sumarização dos dados para envio ao *Middleware* do Bairro foi realizada no *Middleware* da Rua 1. Por esse motivo, para fins comparativos de desempenho de processamento, utilizou-se o mesmo Raspberry da Rua 1 como o *Middleware* para o presente caso.

A Figura 4.6 mostra o uso de CPU pelo *Raspberry* que roda o cliente. O comportamento médio mostra um uso em torno de 30% e picos, onde de valor mais alto foi de 33%.

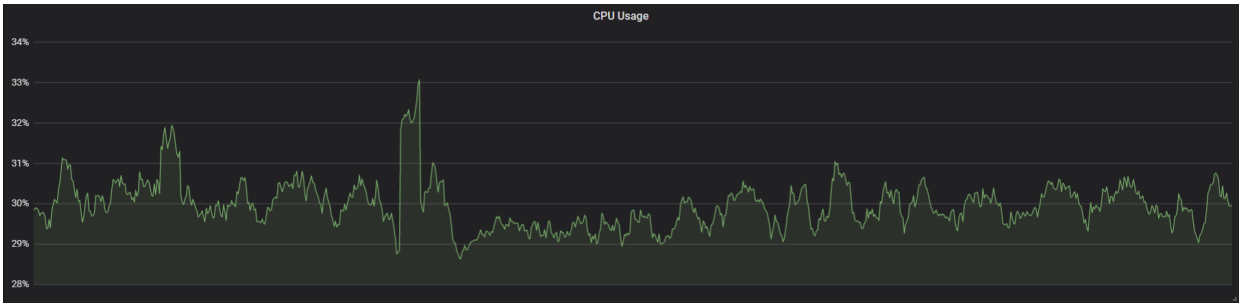

Figura 4.6 – Uso de CPU dos Clientes - Arquitetura *Cloud*
Fonte: Elaboração dos autores.

A Figura 4.7 mostra o uso de CPU pelo *Middleware*. O uso de CPU está separado por *conteiners* do *Docker*: dims(verde), *engine*(amarelo) e *gateway*(azul). O uso pelo *gateway* permaneceu acima

de 32,5% e abaixo de 35%. O uso pelo *engine* foi entre 25% e 27,5%. O uso pelo dims foi abaixo de 25% e acima de 22,5%.

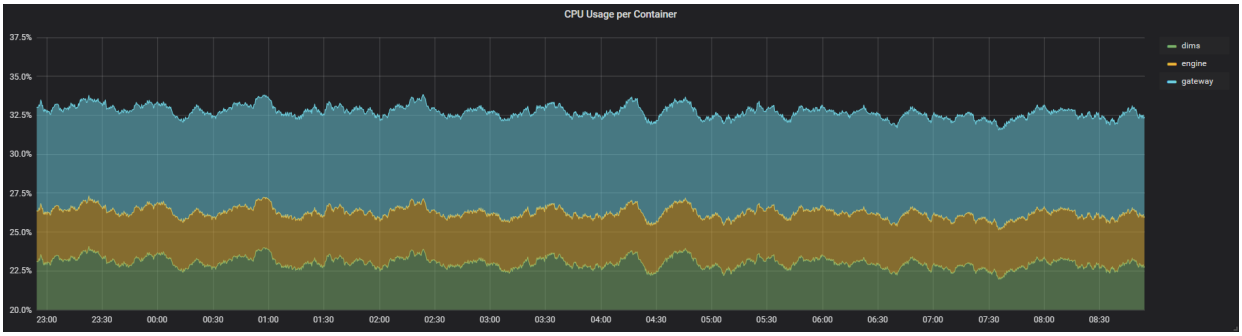

Figura 4.7 – Uso de CPU - Arquitetura *Cloud*
Elaboração dos autores.

4.5 Tráfego de Rede

4.5.1 Arquitetura Hierárquica

Nos gráficos presentes nas imagens desta seção, a cor vermelha representa a taxa de envio de dados de cada *Middleware* ao longo do tempo de execução do caso da Arquitetura Hierárquica. A cor verde representa a taxa de dados recebida por cada *Middleware*.

A Figura 4.8 mostra o comportamento da taxa de dados enviada e recebida ao longo do tempo pelo *Middlaware* que representa a Casa 1.

Observa-se que a taxa de recebimento permaneceu constante. A taxa de envio possui picos que representam o momento de ocorrência dos eventos configurados para enviar dados.

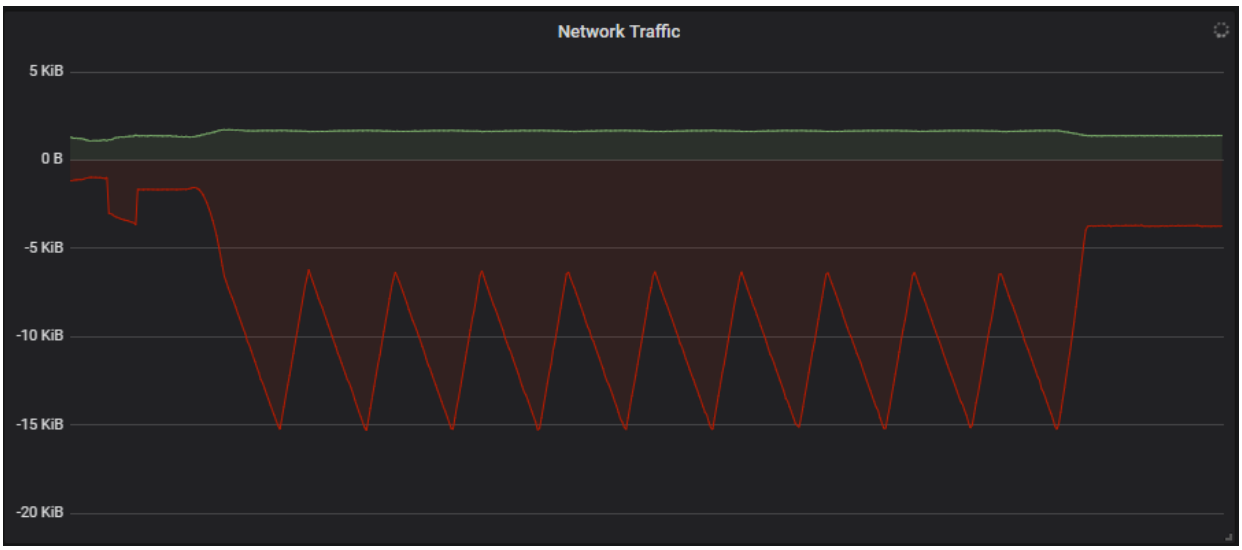

Figura 4.8 – Tráfego de Rede Casa 1 - Arquitetura Hierárquica
Fonte: Elaboração dos autores.

A Figura 4.9 mostra o comportamento da taxa de dados enviada e recebida ao longo do tempo pelo *Middlaware* que representa a Rua 1.

Pode-se notar que, de modo geral, a taxa de recebimento permaneceu constante. A taxa de envio também possui picos que representam o momento de ocorrência dos eventos configurados para enviar dados.

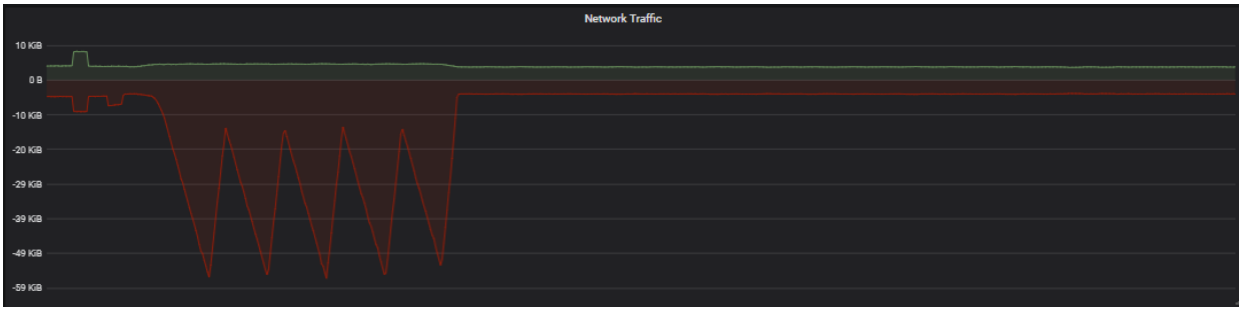

Figura 4.9 – Tráfego de Rede Rua 1 - Arquitetura Hierárquica
Fonte: Elaboração dos autores.

Na Figura 4.10 aparece o comportamento da taxa de dados enviada e recebida ao longo do tempo pelo *Middlaware* que representa o bairro.

Nesse caso a taxa de envio e recebimento são constantes.

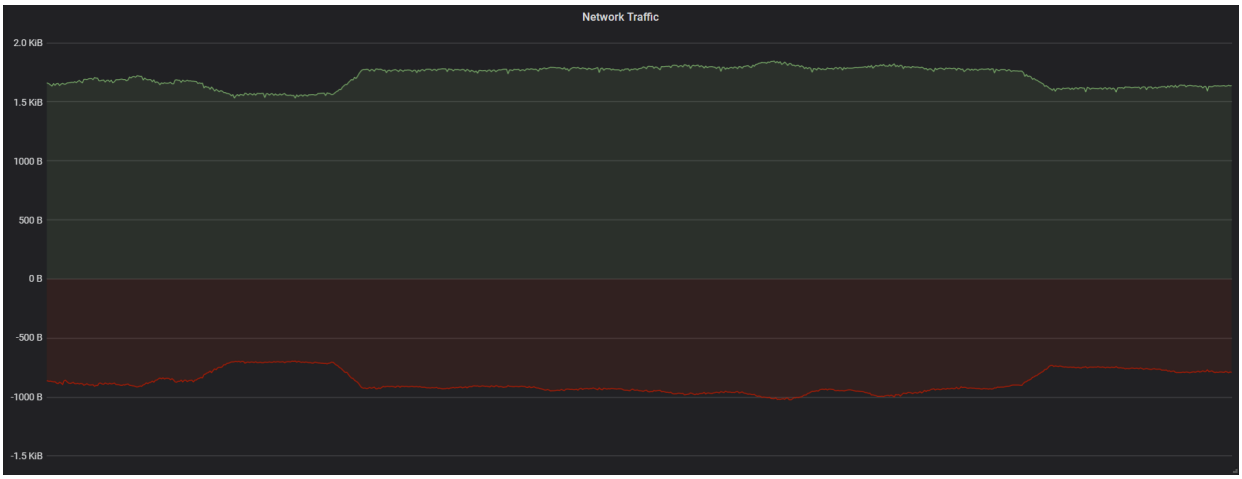

Figura 4.10 – Tráfego de Rede Bairro - Arquitetura Hierárquica
Fonte: Elaboração dos autores.

4.5.2 Arquitetura *Cloud*

Nos gráficos apresentados nesta seção, a cor vermelha representa a taxa de envio de dados de cada *Middleware* ao longo do tempo de execução do caso da arquitetura *cloud*. A cor verde representa a taxa de dados recebida por cada *Middleware*.

A Figura 4.11 representa o tráfego de rede no *Raspberry Pi* que executa o cliente. Aqui observa-se que a taxa de recebimento é constante.

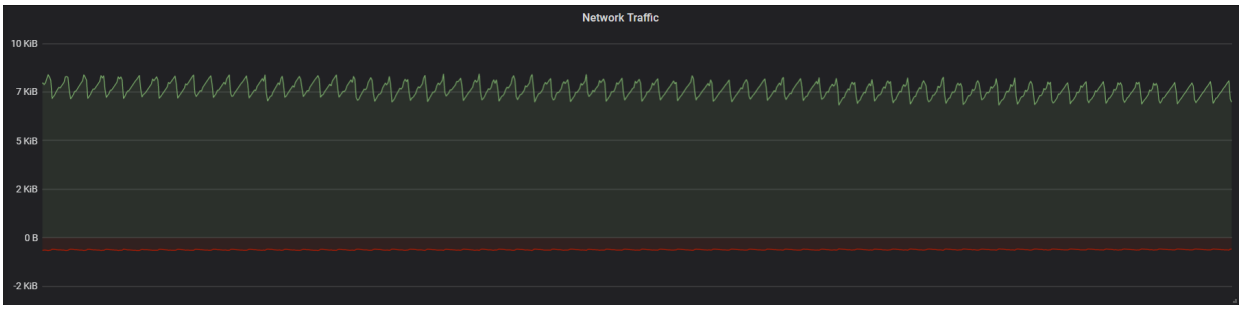

Figura 4.11 – Tráfego de Rede dos Clientes - Arquitetura *Cloud*
Fonte: Elaboração dos autores.

Na Figura 4.12 está presente o tráfego de rede no *Middleware* deste caso. A taxa de envio é constante bem como a taxa de recebimento.

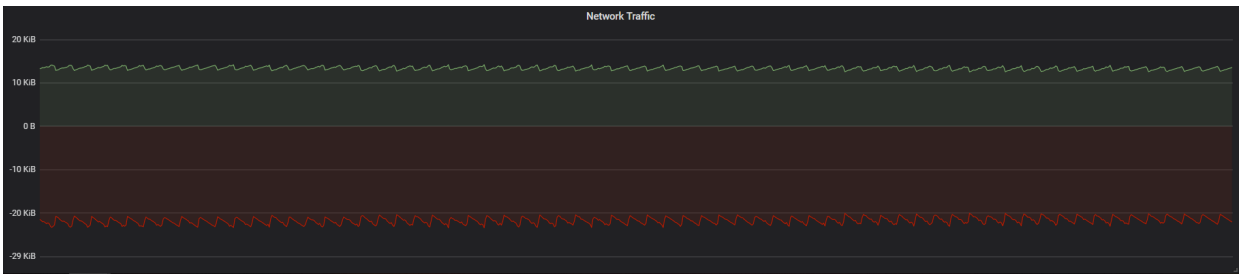

Figura 4.12 – Tráfego de Rede - Arquitetura *Cloud*
Elaboração dos autores.

4.6 Comparativo Arquitetura Hierárquica e Arquitetura *Cloud*

No caso da Arquitetura Hierárquica, a taxa de envio dos *Middlewares* que possuem sensores conectados (Casas e Rua), é composta por picos que acontecem nos momentos de ocorrência dos eventos para envio de dados. Na maior parte do tempo, a taxa permanece constante e baixa já que não é necessário enviar todas as leituras dos sensores para outros *Middlewares*. Assim, o *Middleware* do Bairro recebe poucos dados. A taxa de recebimento nestes *Middlewares* resumem-se a taxa de envio de apenas dispositivos conectados a eles e de dados sumarizados dos outros *Middlewares*.

Já no caso da arquitetura *cloud*, o *Middleware* possui uma taxa alta e constante de envio e recebimento de dados. Isso ocorre, pois há uma centralização do destino dos dados gerados pelos dispositivos e, também, todos os clientes e aplicações que desejam extrair informações dos dados coletados configuram os serviços nesse mesmo nó.

Com o intuito de quantificar e comparar os dados trocados nos dois casos, restringiu-se o período de análise apenas para quando todos os serviços estavam em operação. Isso resultou em

duas horas de execução dois cenários. Assim a quantidade de dados produzida nos dois cenários no período citado é a mesma (Tabelas 4.1, 4.2 e Sensor de Luminosidade na Tabela 4.3 versus dados da Tabela 4.5).

As Tabelas 4.1, 4.2, 4.3 e 4.4 mostram a quantidade de dados enviadas para cada *Middleware* no caso da Arquitetura Hierárquica. Os dados trocados entre *Middlewares* das casas com a Rua somam 9,182 kB. O *Middleware* da Rua envia 4,718 kB para o Bairro. Assim, a soma dos dados trocados entre *Middlewares* no caso da Arquitetura Hierárquica, que são os dados trafegados pela Internet, somam 13,900 kB. Tratando-se dos dados de luminosidade enviados ao *Middleware* Rua, pode-se considerar que esses dados trafegam pela Internet devido a posição geográfica. Isso resulta em 2.158,031 kB.

Tabela 4.1 – Tabela Origem e Quantidade de Dados Enviados para Casa 1 - Arquitetura Hierárquica

Origem	**Quantidade de Dados**
Estação Meteorológica	382,255 kB

Fonte: Elaboração dos autores.

Tabela 4.2 – Tabela Origem e Quantidade de Dados Enviados para Casa 2 - Arquitetura Hierárquica

Origem	**Quantidade de Dados**
Estação Meteorológica	563,968 kB

Fonte: Elaboração dos autores.

Tabela 4.3 – Tabela Origem e Quantidade de Dados Enviados para Rua 1 - Arquitetura Hierárquica

Origem	**Quantidade de Dados**
Casa 1	5,010 kB
Casa 2	4,172 kB
Sensor de Luminosidade	2.144,131 kB
Total	2.153,313 kB

Fonte: Elaboração dos autores.

Tabela 4.4 – Tabela Origem e Quantidade de Dados Enviados para Bairro - Arquitetura Hierárquica

Origem	**Quantidade de Dados**
Rua 1	4,718 kB

Fonte: Elaboração dos autores.

Na Tabela 4.5 está presente a quantidade de dados enviados para o *Middlaware* do caso da arquitetura *cloud*. O valor total é de 3.092,746 kB. E toda essa quantidade de dados seria enviado pela Internet.

Tabela 4.5 – Tabela Origem e Quantidade de Dados Enviados para *cloud*

Origem	**Quantidade de Dados**
Estação Meteorológica 1	377,838 kB
Estação Meteorológica 2	559,522 kB
Sensor de Luminosidade	2.155,386 kB
Total	3.092,746 kB

Fonte: Elaboração dos autores.

Comparando os dois casos, observa-se que, no cenário da arquitetura hierárquica, é enviada para Internet (bairro) uma quantidade de dados 99,55% menor do que quantidade de dados enviados para a *cloud* centralizada, para extrair as mesmas informações. Considerando que os dados enviados a Rua também transitam pela internet no caso da Arquitetura Hierárquica, obtém-se 2.158,031 kB, resultando em uma redução de 69,78% de dados trafegados pela Internet quando comparados a arquitetura em *cloud*. É importante ressaltar que o sensor de luminosidade é responsável pela geração de cerca de 56% dos dados sensoriais de todo o sistema. Isso também reflete no uso de CPU para processamento dos dados. É intuitivo pensar que, no cenário da arquitetura *cloud*, onde está tudo centralizado, haverá o uso de mais recurso de CPU do *hardware* utilizado. Isso pode ser visto atentando-se para os resultados apresentados seção 4.4. Neste momento exploramos benefícios do conceito de *Edge Computing*.

5 Conclusão

Nesse projeto foi desenvolvido um *Middleware* IoT genérico capaz de receber, armazenar e processar dados enviados por dispositivos. Todos os módulos, bem como as funcionalidades de comunicação entre eles, foram implementados com o intuito de obter um sistema escalável e genérico para atender objetivos e arquiteturas diferentes. Na presente pesquisa, foi feita a implantação de uma arquitetura de rede hierárquica utilizando esse sistema.

Foi construída, também, uma forma de abstração das entidades da rede IoT, separando entidades chave para a identificação dos componentes das camadas inferiores da rede IoT. Além disso, desenvolveu-se uma biblioteca para dispositivos Arduino e ESP que constroem as requisições, seguindo as necessidades do sistema em diferentes protocolos de comunicação.

A partir dessa biblioteca, foi possível alimentar o sistema com dados a partir de dispositivos ESP-8266. Desenvolveu-se um projeto e código fonte para o uso do dispositivo com baixo consumo de energia.

Os dados gerados pelos dispositivos construídos com a biblioteca implementada permitiram a comparação de uma arquitetura hierarquizada com *Middlewares* e uma de *cloud* centralizada. Os dados obtidos a partir dos cenários de testes propostos e executados permitem observar os seguintes pontos em cada métrica:

- Comportamento do Tráfego de Rede: No cenário com *Cloudlets*, os *Middlewares* possuem picos pontuais na taxa de envio e recebimento. Em contra partida, no cenário com uma *cloud* centralizada, o *Middleware* possui contantes taxas de envio e recebimento de dados. O tempo ocioso do sistema sobre o uso de banda para o envio de dados se mostra positivo, visto que o canal poderá ser utilizado para outras aplicações, sem necessidade de competição sobre ele.
- Uso de CPU: Na arquitetura hierárquica, o processamento na borda da rede mostrou demandar menos recurso de CPU dos *hardwares* utilizados comparados ao caso com uma *cloud* centralizada, além de explorar os benefícios de processamento distribuído de informações, que permitem a melhor distribuição de recursos na *cloud* para aplicações que apenas podem ser processadas nesta.
- Quantidade de Dados Trocados pela Internet: Na análise quantitativa de dados gerados que são trafegados pela Internet, o uso de *Middlewares* distribuídos em uma arquitetura em árvore se mostrou que é possível extrair as mesmas informações reduzindo o tráfego na rede.

A hipótese de que uma arquitetura hierárquica diminuiria a quantidade de dados trafegados pela Internet foi confirmada a partir da análise exposta.

O sistema implementado, por possuir ferramentas de gerenciamento de eventos generalizado para os dados enviados e ser configurável para envio a diferentes destinos é facilmente implan-

tável em qualquer arquitetura para variados objetivos. O mecanismo também provê um maior empoderamento sobre a rede de internet das coisas gerido pelo sistema.

A confirmação da hipótese, juntamente com os sistemas desenvolvidos, atenderam todos os objetivos propostos da pesquisa.

5.1 Trabalhos Futuros

Na pesquisa executada não foram implementadas conexões seguras entre dispositivos e nem entre *Middlewares*. Levando em consideração a sensibilidade dos possíveis dados coletados, é muito importante a aprofundação no tema e desenvolvimento de um modelo seguro que provê integridade, confidencialidade e disponibilidade dos serviços e informações.

Pela forma que o *Middleware* foi arquitetado e desenvolvido, o caminho está aberto para implementações de algoritmos de automatização de processamentos e técnicas de *Machine Learning* para auxílio na tomada de decisão relativas as sumarizações de dados. Pela quantidade massiva de dados gerados e trafegados na rede, esforços nesse âmbito é de grande valia.

Pelo escopo do projeto, os *Middlewares* encontravam-se conectados. Permitir que *Cloudlets* configurem outras instâncias que encontram-se em redes privadas diferentes, abriria um leque de possibilidades no que tange a coleta de informações a partir de dados brutos de diferentes escopos.

Bibliografia

[1] I.F. Akyildiz et al. “Wireless sensor networks: a survey”. Em: *Computer Networks* 38 (2002), pp. 393–422.

[2] K ASHTON. “That “Internet of Things” thing”. Em: *RFiD Journal* (2009).

[3] L. Atzori, A. Iera e G. Morabito. “The Internet of Things: a survey”. Em: *Computer Networks 54* (2010), pp. 2787–2805.

[4] Francisco L. de Caldas Filho et al. “Design and Evaluation of a Semantic Gateway Prototype for IoT Networks”. Em: *UCC Companion* (2017).

[5] Francisco L. de Caldas Filho et al. “GERENCIAMENTO DE SERVIÇOS IoT COM GATEWAY SEMÂNTICO”. Em: *Conferências Ibero-Americanas WWW/Internet e Computação Aplicada* (2017).

[6] CISCO. “Cisco Visual Networking Index: Forecast and Trends, 2017–2022”. Em: *White paper Cisco public* (2019).

[7] D. Evan. *The internet of things: How the next evolution of the internet is changing everything.* 2011. URL: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>.

[8] A. Ghosh e S.K. Das. “Coverage and connectivity issues in wireless sensor networks: a survey”. Em: *Pervasive and Mobile Computing* 4 (2008), pp. 303–334.

[9] Jayavardhana GUBBI et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. Em: *Future Generation Computer Systems* 29 (2013), pp. 1645–1660.

[10] Shivayogi HIREMATH, Geng YANG e Kunal MANKODIYA. “Wearable Internet of Things: Concept, Architectural Components and Promises for Person-Centered Healthcare”. Em: *4th International Conference on Wireless Mobile Communication and Healthcare - Transforming Healthcare Through Innovations in Mobile and Wireless Technologies (MOBIHEALTH)*. 2014.

[11] International Telecommunication Union (ITU). “The Internet of Things”. Em: *Genebra: ITU Internet Reports* (2005).

[12] Y. Jararweh et al. “The future of mobile cloud computing: Integrating cloudlets and mobile edge computing”. Em: *23rd Int. Conf. Telecommun. (ICT)* (2016), pp. 1–5.

[13] Juyong Lee e Jihoon Lee. “Hierarchical Mobile Edge Computing Architecture Based on Context Awareness”. Em: *Applied Sciences* 8 (2018), pp. 2076–3417.

[14] J. Lin et al. “A real-time en-route route guidance decision scheme for transportation-based cyberphysical systems”. Em: *IEEE Trans. Veh. Technol.* 66 (2017), pp. 2551–2566.

[15] Lucas M. C. e Martins et al. “Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices”. Em: *UCC Companion* (2017).

[16] N. Mohamed et al. “SmartCityWare: A service-oriented middleware for cloud and fog enabled smart city services”. Em: *IEEE Access* 5 (2017), pp. 17576–17588.

[17] C.S. NANDYALA e H.K KIM. “From Cloud to Fog and IoT-Based Real-Time U-Healthcare Monitoring for Smart Homes and Hospitals”. Em: *International Journal of Smart Home* 10 (2016), pp. 187–196.

[18] A. Rahman, D. Gellert e D. Seed. “A Gateway architecture for interconnecting smart objects to the internet”. Em: *Applied Sciences - Open Access Journal* 25 (2011).

[19] David Reinsel, John Gantz e John Rydning. *The Digitization of the World*. 2018. URL: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>.

[20] K. Rose, S. Eldridge e L. Chapin. “The Internet of Things: An overview”. Em: *Internet Soc. (ISOC)* (2015), pp. 1–53.

[21] Y. Sang et al. “Secure Data Aggregation in Wireless Sensor Networks: A Survey”. Em: *Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Application and Technologies (PDCAT’06)* (2006), pp. 315–320.

[22] M. SATYANARAYANAN et al. “The case for vm-based cloudlets in mobile computing”. Em: *IEEE Pervasive Computing* 99 (2011).

[23] Caio C. M. Silva et al. “Proposta de auto-registro de serviços pelos dispositivos em ambientes de IoT”. Em: *XXXIV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES - SBrT2016* (2016).

[24] H. Sundmaeker et al. *Vision and challenges for realising the Internet of Things*. 2010.

[25] l. TONG, Y LI e W GAO. “A hierarchical Edge cloud architecture for mobile computing”. Em: *IEEE INFOCOM 2016. The 35th Annual International Conference on Computer Communication* (2016).

[26] T. VERBELEN et al. “Cloudlets: Bringing the cloud to the mobile user”. Em: *Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services* (2012).

[27] E. Welbourne et al. “Building the Internet of Things using RFID: The RFID ecosystem experience”. Em: *IEEE Internet Computing* 13 (2009), pp. 48–55.

[28] Y. Yan et al. “A survey on cyber security for smart grid communications”. Em: *IEEE Commun. Surveys Tuts.* 5 (2017), pp. 998–1010.

[29] Wei Yu et al. “A survey on the edge computing for the Internet of Things”. Em: *IEEE Access* 6 (2017), pp. 6900–6919.

ANEXO A – Síntese do Cisco *Visual Networking Index*

O Cisco *Visual Networking Index* – VNI (Cisco, 2019) apresenta previsões para o tráfego mundial total da Internet e outras questões ligadas a temática como previsões sobre o número de dispositivos conectados à Internet de forma quantitativa e sempre crescentes. O referido relatório mostra que esse sofreu um aumento significativo nas últimas duas décadas. Conforme pode ser observado na Figura A.1, em 1992 ele era de aproximadamente 100 GB por dia, em 2002 esse volume cresceu para 100 GB/segundo, e atingiu mais de 45.000 GB/segundo em 2017. A estimativa é que esse volume chegue a 150.700 GB/segundo em 2022 (CISCO, 2019).

Year	Global internet traffic
1992	100 GB per day
1997	100 GB per hour
2002	100 GB per second
2007	2,000 GB per second
2017	46,600 GB per second
2022	150,700 GB per second

Figura A.1 – Cisco VNI previsão: Contexto histórico da internet
Fonte: Cisco, [6].

.

O relatório também apresenta nove tendências associadas, são elas:

- Mudanças contínuas no *mix* de dispositivos e conexões.
- A adoção do IPv6 permite a conectividade da Internet das Coisas (IoT).
- Aplicações M2M[1] em muitas indústrias aceleram o crescimento da IoT.
- Crescimento de tráfego de aplicações.
- Análise "*Cord-Cutting*[2]".
- Análise de segurança.
- Efeitos da aceleração de velocidades no crescimento do tráfego.

[1] M2M - *Machine to Machine, que representam conexões diretas entre dois dispositivos*
[2] Cord-Cutting - Termo utilizado para descrever o cancelamento de pessoas sobre a inscrição em serviços de TV por assinatura

- Mobilidade (Wi-Fi) continua a ganhar impulso.
- Análise do padrão de tráfego (pico em comparação com a média, captação de CDN[3] e SD-WAN[4])

Em seguida, apresenta-se uma síntese dos principais dados apresentados para justificar algumas dessas tendências, associadas ao crescimento do volume de dados na Internet, segundo o relatório da Cisco VNI 2019.

a) Mudanças contínuas no *mix* de dispositivos e conexões

Verifica-se um crescimento global acelerado dos dispositivos e das conexões (10% de CAGR[5]), muito maior do que o crescimento da população mundial (1% de CAGR) e o de usuários de Internet (7% de CAGR).

> A cada ano, vários dispositivos novos em diferentes formatos com capacidade e inteligência aumentadas são introduzidos e adotados no mercado. Um número crescente de aplicativos M2M, como medidores inteligentes, vigilância por vídeo, monitoramento de assistência médica, transporte e rastreamento de pacotes ou ativos, contribuem de maneira importante para o crescimento de dispositivos e conexões. Até 2022, as conexões M2M serão 51% do total de dispositivos e conexões (Cisco, 2019 [7]).

Sendo que, as conexões máquina-máquina (M2M) apresentam-se como a categoria que mais deverão crescer, a uma taxa de quase 2,4 vezes até 2022, apresentado 19% CAGR, com a estimativa de chegar a 14,6 bilhões de conexões até o final do período. Eles serão seguidos pelos *smartphones* que apresentam um CAGR de 9% e pelas TVs conectadas com 7% de CAGR. Apesar de, ao longo do período, os computadores pessoais (PC[6]) apresentarem declínio de 2,5%, eles ainda serão maioria em relação aos *tablets* (1,2 bilhão de PCs contra 790 milhões de *tablets*) – Figura A.2.

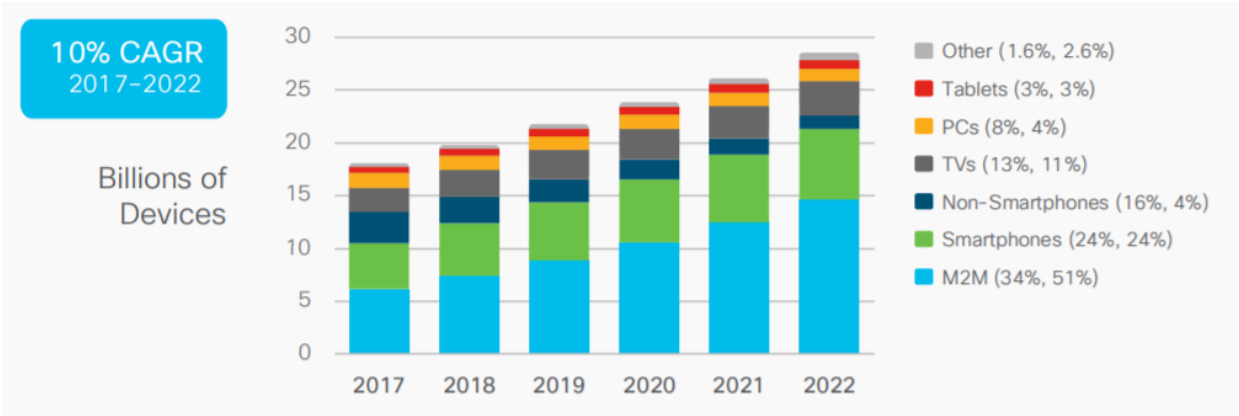

Figura A.2 – Crescimento global de dispositivos e conexões
Fonte: Cisco, [6].

.

[3] CDN - *Content Delivery Network, que são redes de servidores de proxy* geograficamente distribuidos para atender a demanda de um conteudo na Internet
[4] SD-WAN - Software Defined Wide-Area Network
[5] CAGR - *Compound Annual Growth Rate*
[6] PC - Personal Computer

Segundo o *Visual Networking Index* da Cisco, a mudança no *mix* de dispositivos e conexões e o crescimento de múltiplos dispositivos afetarão o tráfego na rede. Conforme avaliado nos gráficos expostos nas Figuras A.3 e A.4, no final de 2017, 59% do tráfego IP e 51% do tráfego da Internet originaram-se de dispositivos que não eram PC e até 2022, haverá um crescimento desse comportamento para 81% do tráfego IP[7] e do tráfego da Internet [8]. Destaca-se que no caso das redes móveis os dispositivos de vídeo deverão apresentar efeito multiplicador no tráfego, em função dos usuários migrarem a visualização de vídeos da TV HD[9] para *smartphones* e *tablets*. Os *smartphones* representarão 50% do total do tráfego global da Internet até 2022, em comparação com 23% em 2017.

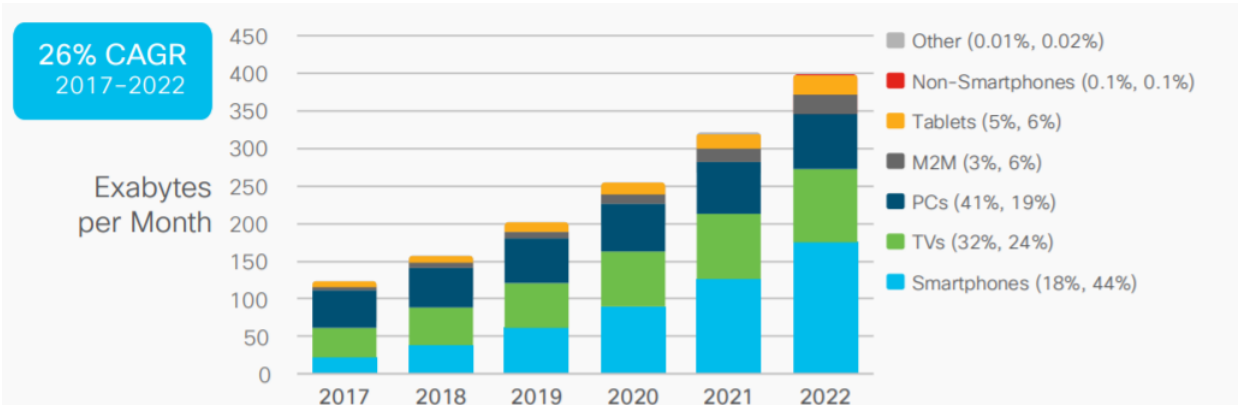

Figura A.3 – Tráfego de IP global por dispositivo
Fonte: Cisco, [6].

.

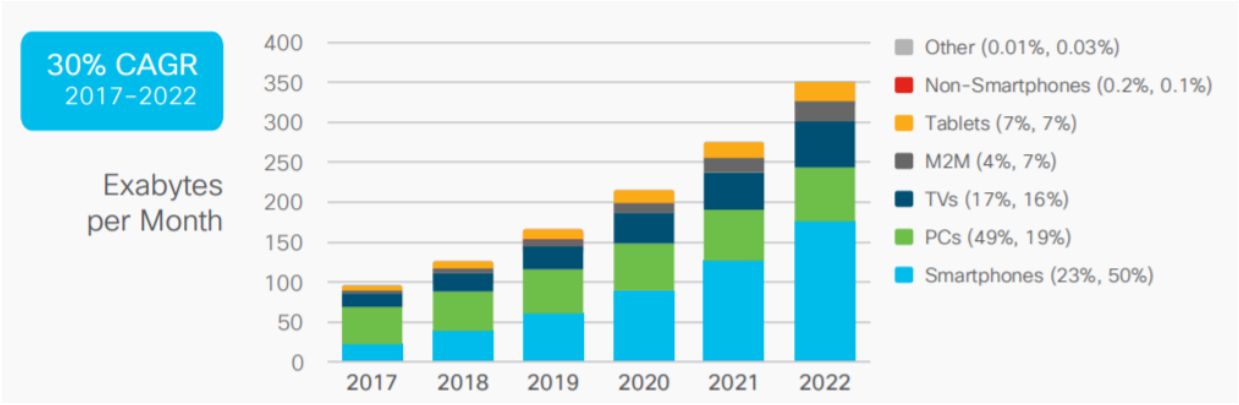

Figura A.4 – Tráfego global da internet por tipo de dispositivo
Fonte: Cisco, [6].

.

b) Aplicações M2M em muitas indústrias acelerarão o crescimento da IoT

[7] Tráfego IP - Tráfego gerenciado de uma origem a um destino pelo mesmo *service provider*, passando apenas por uma rede
[8] Tráfego da Internet - Inclui todo tráfego IP que passa por um *backbone* na Internet
[9] HD - High Definition

A evolução da IoT é resultado do crescimento das conexões M2M, que deverão crescer 2,4 vezes, de 6,1 bilhões em 2017 para 14,6 bilhões até 2022 (Figura A.5). A estimativa é de que haverá 1,8 conexões M2M para cada pessoa no mundo até 2022.

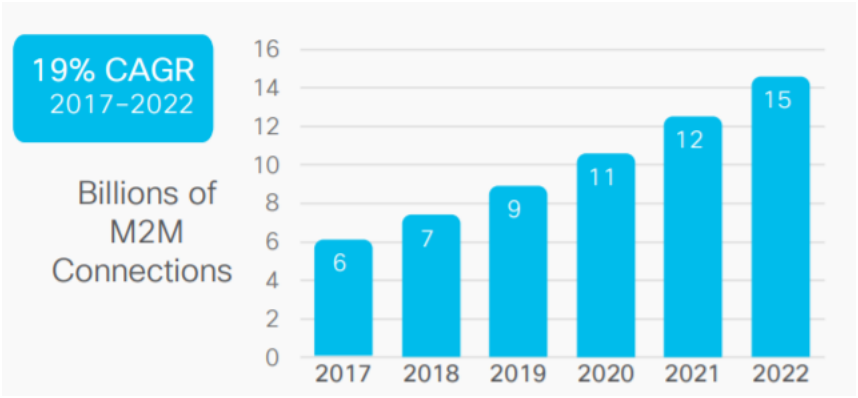

Figura A.5 – Crescimento das conexões M2M globais
Fonte: Cisco, [6].

.

Por outro lado, o tráfego IP M2M global crescerá mais de sete vezes no mesmo período, de 3,7 EB por mês em 2017 (3% do tráfego IP global) para mais de 25 EB em 2022 (6% de tráfego IP global, consulte a Figura A.6). Cabe destacar que a quantidade de tráfego vem crescendo mais rápido do que o número de conexões. Isso vem ocorrendo, pois há um aumento da implantação de aplicativos de vídeos em conexões M2M, bem como o aumento do uso de aplicativos, como por exemplo os voltados para a telemedicina, sistemas de navegação inteligente. Todos eles exigindo maior largura de banda e menor latência.

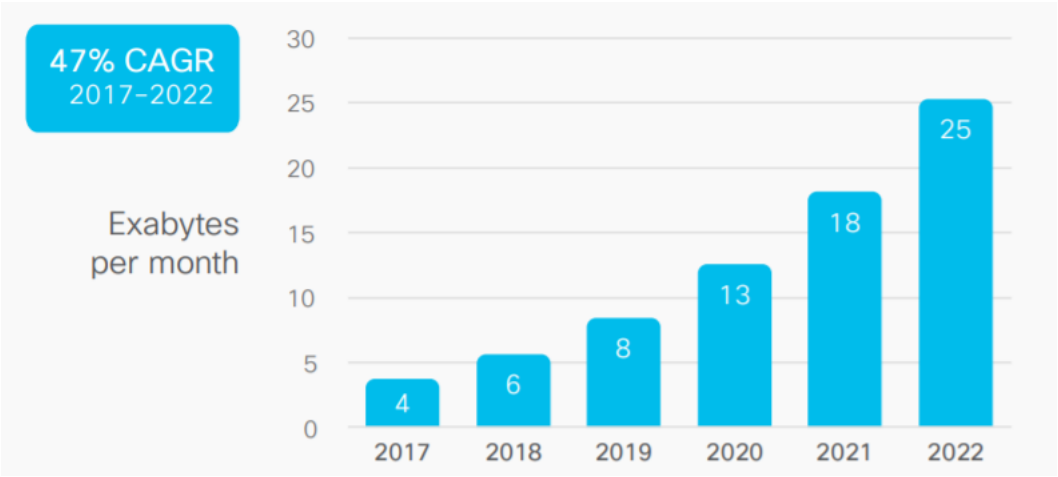

Figura A.6 – Crescimento do tráfego global de M2M
Fonte: Cisco, [6].

.

c) Crescimento de tráfego de Aplicações

Globalmente, o tráfego de vídeo IP, em todas as suas formas – vídeo da Internet, IP VoD[10], arquivos de vídeo trocados por compartilhamento de arquivos, videogames e videoconferência – será responsável por 82% do tráfego até 2022 (Figura A.7), apresentando um padrão de tráfego dinâmico.

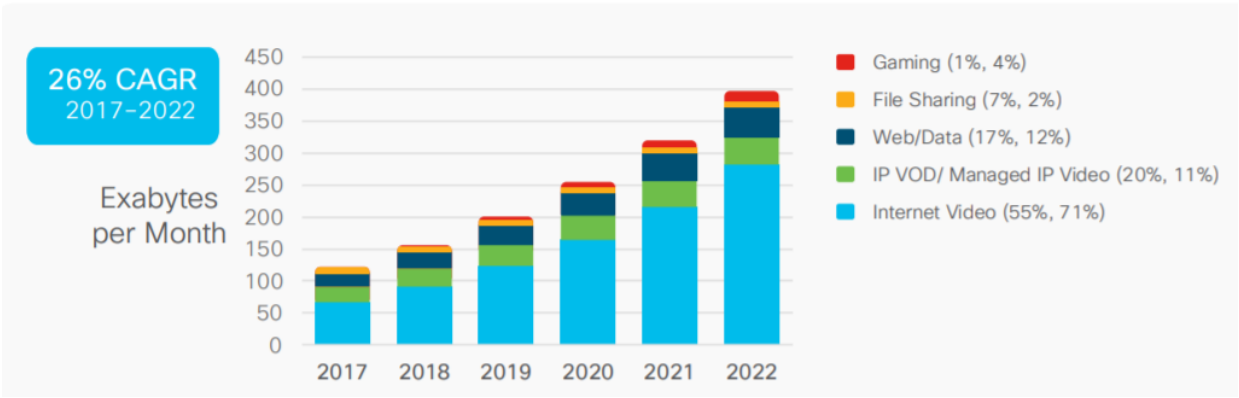

Figura A.7 – Tráfego de IP global por categorias de aplicações
Fonte: Cisco, [6].

.

Verificou-se nos últimos anos, o aumento no tráfego associado a *downloads* de jogos nos provedores de serviços, em função das características do consoles mais novos, como o *Xbox One* e o *PlayStation* 4, apresentarem armazenamento interno suficiente para permitir que os jogadores baixem novos jogos em vez de comprá-los em disco, representando 4% de todo o tráfego IP até 2022.

[10] VoD - Video on Demand