GATEWAY REDUNDANTE IOT COM USO DE FOG COMPUTING

CASSIO FABIUS CAMBRAIA RIBEIRO

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GATEWAY REDUNDANTE IOT COM USO DE FOG COMPUTING

CASSIO FABIUS CAMBRAIA RIBEIRO

Orientador: RAFAEL TIMÓTEO DE SOUSA JÚNIOR, ENE/UNB

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO

PUBLICAÇÃO PPGENE.DM - XXX/AAAA
BRASÍLIA-DF, 15 DE FEVEREIRO DE 2019.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GATEWAY REDUNDANTE IOT COM USO DE FOG COMPUTING

CASSIO FABIUS CAMBRAIA RIBEIRO

MONOGRAFIA DE GRADUAÇÃO ACADÊMICA SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA DE REDES DE COMUNICAÇÃO.

APROVADA POR:

Rafael Timóteo de Sousa Júnior, ENE/UnB
Orientador

Examinador Interno
Georges Daniel Amvame Nze, ENE/UnB

Francisco Lopes de Caldas Filho, ENE/UnB
Coorientador e examinador interno

Robson de Oliveira Albuquerque, Gabinete de Segurança Institucional
Examinador Externo

BRASÍLIA, 15 DE FEVEREIRO DE 2019.

FICHA CATALOGRÁFICA
CASSIO FABIUS CAMBRAIA RIBEIRO
Gateway Redundante IoT com uso de Fog Computing
2019xv, 60p., 201x297 mm
(ENE/FT/UnB, Bacharel, Engenharia de Redes de Comunicação, 2019)
Monografia de Graduação - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

CASSIO FABIUS CAMBRAIA RIBEIRO (2019) Gateway Redundante IoT com uso de Fog Computing. Monografia de Graduação em Engenharia de Redes de Comunicação, Publicação xxx/AAAA, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 60p.

CESSÃO DE DIREITOS

AUTOR: CASSIO FABIUS CAMBRAIA RIBEIRO
TÍTULO: Gateway Redundante IoT com uso de Fog Computing.
GRAU: Bacharel ANO: 2019

_____
CASSIO FABIUS CAMBRAIA RIBEIRO

Agradecimentos

Agradeço primeiramente à minha família, principalmente aos meus pais e à minha avó, por todo apoio incondicional que me deram durante todos esses anos. Agradeço ao meu irmão e aos meus amigos pelo companheirismo e pelos momentos de descontração e incentivo.

Agradeço à Fernanda Xavier de Araújo pelo amor e carinho que me deu, por ser a pessoa que sempre está ao meu lado para me ajudar a passar pelos momentos mais difíceis, além dos conselhos que me ajudaram a me encontrar tanto acadêmica quanto profissionalmente.

Agradeço a todos os membros do Laboratório UIoT, especialmente ao Francisco Lopes de Caldas Filho e ao Lucas Maurício Castro e Martins, por toda ajuda prestada e por todo conhecimento passado a mim nos últimos anos. Boa parte do que aprendi nesses anos de universidade foi junto de vocês e por causa de vocês.

O Laboratório UIoT conta com recursos de apoio das Agências brasileiras de pesquisa, desenvolvimento e inovação CAPES (Projeto FORTE 23038.007604/2014-69), CNPq (Projeto INCT em Segurança Cibernética 465741/2014-2) e Fundação de Apoio à Pesquisa do Distrito Federal FAPDF (Projetos UIoT 0193.001366/2016 e SSDDC 0193.001365/2016), bem como do Gabinete de Segurança Institucional da Presidência da República (TED 002/2017) e do Laboratório LATITUDE/UnB (Projeto SDN 23106.099441/2016-43).

Por fim, agradeço também ao Georges Daniel Amvame Nze, por ter sido o professor que me fez decidir mudar de curso após um semestre de aula sobre redes, e ao Rafael Timóteo de Sousa Júnior, pelo incentivo ao crescimento acadêmico que tive por conta do UIoT, dos cursos, dos congressos e das oportunidades que me deu.

Obrigado de coração a todos vocês.

CASSIO FABIUS CAMBRAIA RIBEIRO

RESUMO

A Internet das Coisas (IoT) está se popularizando cada vez mais, tanto nas soluções industriais quanto nas residenciais. Em conjunto com a computação ubíqua, vários objetos do cotidiano estão ganhando poder de processamento e formas de se comunicar entre si, com a rede local e com a Internet. *Smartphones*, *smartbands*, *smartwatches* e qualquer outro dispositivo "inteligente", como geladeiras que notificam o usuário sobre algum produto que acabou ou quando o filtro deve ser trocado, estão cada vez mais comuns e acabam facilitando e modificando a rotina das pessoas sem que percebam.

Dispositivos IoT em ambientes mais críticos, como *Data Centers*, fábricas e hospitais, estão rapidamente se espalhando e se tornando gradativamente mais necessários para os processos, monitoramentos e manutenções de sistemas e ambientes, sendo possível acompanhar remotamente e praticamente em tempo real suas características e ainda prever falhas e problemas que ainda estão para acontecer. *Big Data* e *Cloud Computing* são conceitos bem íntimos de Internet das Coisas e de manutenção preventiva, mas certas ocasiões exigem uma resposta rápida do *middleware*, quando um determinado atuador precisa ser acionado, por exemplo. Se esse *middleware* estiver hospedado na nuvem, a latência pode ser um problema.

Além disso, redes IoT costumam ser heterogêneas, sendo compostas de dezenas ou até centenas de dispositivos com diferentes atributos, como poder variável de processamento e armazenamento e diversas maneiras de se comunicar. Para permitir que todos esses dispositivos se comuniquem entre si e com a rede, é comum utilizar um equipamento chamado *gateway* IoT, que pode receber e enviar dados utilizando vários protocolos de comunicação, como HTTP, MQTT, *Bluetooth* e *Zigbee*.

Por ser um elemento de tamanha importância na rede, o gateway IoT acaba se tornando um ponto único de falha em ambientes críticos. Soluções para esse problema incluem a aplicação de redundância, garantindo alta disponibilidade do *gateway* IoT. Para reduzir a latência e utilizar recursos ociosos disponíveis após a implementação do *gateway* IoT redundante, também é possível aplicar a arquitetura de *Fog Computing*, estendendo a *Cloud Computing* para a borda da rede.

Esse trabalho apresentará uma proposta que ataca diretamente a vulnerabilidade do *gateway* IoT sendo um ponto único de falha utilizando a redundância e a *Fog Computing* para trazer mais benefícios a uma rede IoT.

Palavras-Chave: Gateway, Redundância, Fog Computing, Internet das Coisas, Protocolos de Comunicação

ABSTRACT

The Internet of Things is becoming increasingly popular in both the industrial and residential solutions. In conjunction with ubiquitous computing, everyday objects are gaining processing power and ways of communicating with each other, with the local network and with the Internet. Smartphones, smartbands, smartwatches and any other "smart"device, such as refrigerators that notify the user that a product has run out or when the filter has to be changed, are becoming more commonplace and end up making people's routine easier and different.

IoT devices in critical environments such as data centers, factories and hospitals are rapidly spreading and becoming increasingly necessary for processes, monitoring and maintenance of systems and environments. It is possible to remotely monitor their characteristics practically in real time and to predict failures and problems that have not occurred yet. Big Data and Cloud Computing are very related concepts to Internet of Things and preventive maintenance, but certain occasions require a quick response from the middleware when a certain actuator needs to be triggered, for example. If this middleware is hosted in the cloud, then latency can be a problem.

In addition, IoT networks are often heterogeneous, consisting of dozens or even hundreds of devices with different attributes, such as processing and storage variable power and various ways of communicating. In order to allow all these devices to communicate with each other, it is common to use a device called IoT gateway, which can receive and send data using various communication protocols, such as HTTP, MQTT, Bluetooth and Zigbee.

Because it is such an important network element, the IoT gateway becomes a single point of failure in critical environments. Solutions to this problem include implementation of redundancy, ensuring high availability of the IoT gateway. To reduce latency and utilize idle resources available after the insertion of a redundant IoT gateway, it is also possible to apply the Fog Computing architecture by extending Cloud Computing to the edge of the network.

This paper will present a proposal that directly addresses the vulnerability to IoT gateway as a single point of failure using redundancy and Fog Computing to bring more benefits to an IoT network.

Keywords: Gateway, Redundancy, Fog Computing, Internet of Things, Communication Protocols

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

Neste momento, a Internet das Coisas (IoT) está em seu auge. De acordo com a empresa Gartner (GARTNER, INC., 2017), o número de dispositivos IoT conectados no mundo ultrapassou a quantidade de pessoas existentes em algum momento de 2017. Essa é uma estimativa conservadora, já que outras pesquisas, como a da IHS Markit (IHS MARKIT), afirmam que esse número era muito superior no mesmo ano, chegando a 27 bilhões.

Por conta da popularização de novos *gadgets* e equipamentos ditos "inteligentes", do desenvolvimento de novas tecnologias e da redução de custo das antigas, o número de dispositivos IoT está crescendo exponencialmente, conforme dados apresentados pela empresa Cisco (STACK, 2018) e por outras pesquisas. Na figura 1.1, há um gráfico mostrando uma previsão de que haverá mais de 75 bilhões de dispositivos IoT conectados espalhados pelo mundo no ano de 2025.

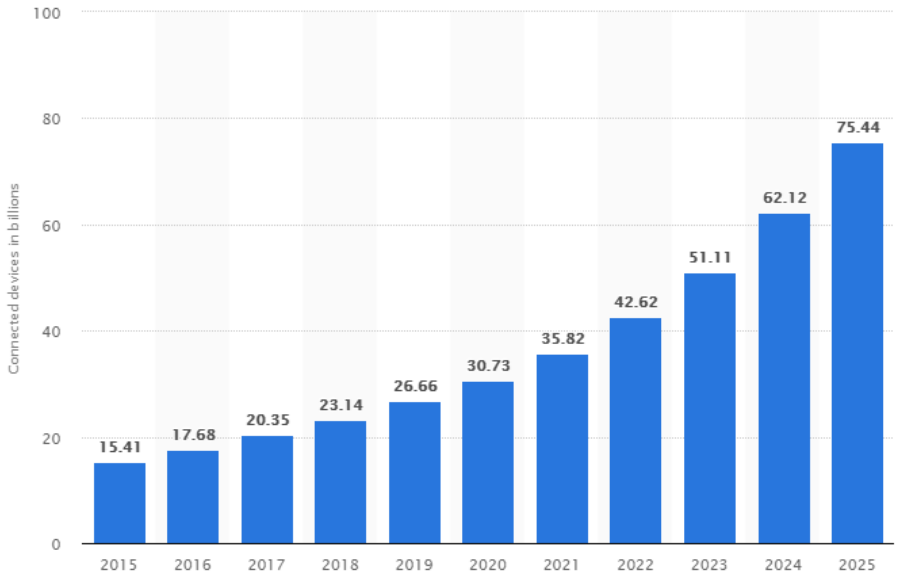

Figura 1.1: Número previsto de dispositivos conectados pelo mundo entre 2015 e 2025 (STATISTA, 2016)

Além do número de dispositivos estar aumentando, a quantidade de dados que estão sendo gerados também é gradativamente maior (figura 1.2). Isso consequentemente gerará uma necessidade maior de banda para a comunicação com a nuvem, onde geralmente os da-

dos são armazenados e analisados, baseando-se na arquitetura conhecida como *Cloud Computing*.

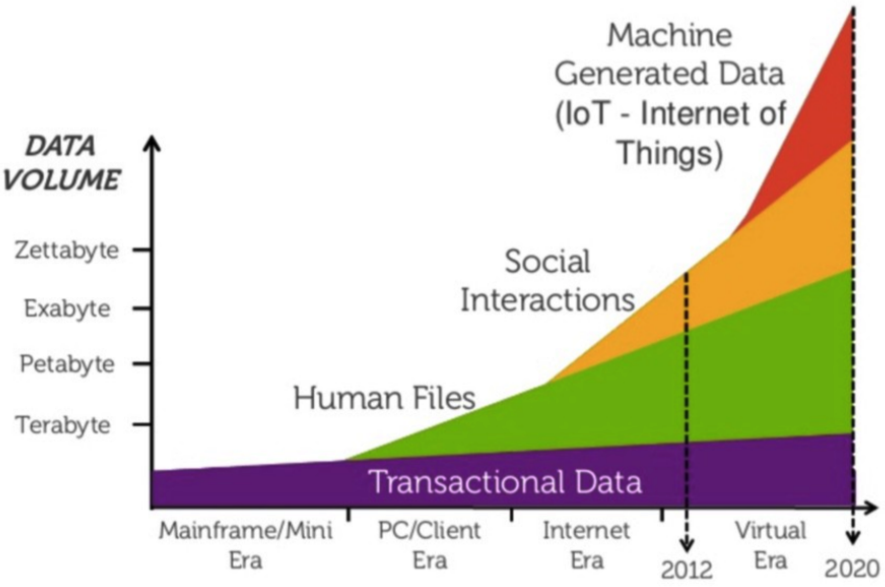

Figura 1.2: Evolução do volume de dados gerados por ano (PANDIT, 2018)

Com literalmente bilhões de coisas enviando e recebendo dados pelo mundo, os recursos de redes estão sendo cada vez mais consumidos pela chamada comunicação máquina a máquina (M2M), gerando certa preocupação (DQE COMMUNICATIONS, 2018). Soluções para possíveis problemas que a Internet das Coisas pode trazer para a sociedade já estão sendo estudadas e desenvolvidas tanto na indústria quanto na academia.

1.1 Definição dos Problemas

Em ambientes domésticos ou industriais repletos de dispositivos IoT que utilizam protocolos de comunicação heterogêneos para se comunicar pela rede, a presença de um *gateway* IoT é essencial (EMBITEL, 2017). Por ser um elemento com tanta importância, mas geralmente único, ele acaba se tornando um ponto vulnerável em caso de falha, principalmente na indústria, onde dados costumam ser críticos e mesmo pequenas perdas podem impactar cadeias de produção, análises e monitoramentos, sendo capazes de gerar grandes prejuízos econômicos (GARTNER, INC., 2014).

Uma possível solução para esse problema é criar uma redundância nesse ponto, o que proveria alta disponibilidade e a possibilidade de realizar manutenções e melhorias no sistema de maneira que provoque o mínimo possível de prejuízo e perda de dados. Inserir uma redundância física costuma ser caro, mas se o sistema for sensível a períodos de indisponibilidade, o investimento pode valer a pena (NATIONAL INSTRUMENTS CORPORATION, 2008). Infelizmente, a redundância cria outros inconvenientes: o dispositivo adicional disponibilizado para assumir o posto do *gateway* que falhou fica subutilizado e gasta energia desnecessariamente enquanto permanece em espera.

Para resolver, ou ao menos minimizar, esse novo revés, há outros recursos que podem ser

implementados, como modos de suspensão de energia, balanceamento de carga para uso de recursos ociosos ou atribuição de novas funções para o dispositivo em espera de uma falha no dispositivo primário.

Outrossim, para ambientes IoT, uma arquitetura baseada em *Fog Computing* é geralmente melhor que *Cloud Computing* (NWN, 2018). Com dispositivos capazes de receber, analisar e enviar dados a poucos saltos dos dispositivos IoT finais, há maior velocidade no acionamento de atuadores, economia de banda ao enviar dados mais condensados à nuvem e certo grau de independência da Internet. Este último benefício é especialmente útil, já que atuadores podem permanecer travados em um certo estado caso não tenham acesso a redes externas, como uma lâmpada que só poderia ser ligada através de uma conexão passando pela Internet, se comunicando com um *middleware* hospedado na nuvem.

Com isso, os problemas podem ser definidos como:

1. O *gateway* é um ponto único de falha da rede IoT;
2. A redundância nesse ponto pode ser cara e ineficiente;
3. *Cloud Computing* é um paradigma pior que *Fog Computing* para redes IoT;

1.2 Objetivos

Após a análise dos problemas a serem enfrentados e a ponderação das possíveis soluções a serem aplicadas, o objetivo principal para o presente trabalho é definido como sendo o desenvolvimento de uma arquitetura baseada em *Fog Computing* para *gateway* IoT que ofereça alta disponibilidade e que permita a distribuição da carga de processamento dos dados do ambiente.

Para isso, pode-se dividir o projeto nos seguintes passos:

1. **Redundância**: criar uma redundância para um *gateway* IoT, reduzindo, dessa maneira, o tempo de indisponibilidade que dispositivos IoT com sensores e atuadores sofrerão em caso de falha no *gateway* primário da rede.
2. ***Fog Computing***: utilizar um *middleware* IoT para estender a computação da nuvem até a borda da rede, por poder trazer benefícios tanto para os usuários quanto para a rede em si em certas situações (YI; C. LI; Q. LI, 2015);
3. **Uso de recursos ociosos**: aproveitar o *gateway* IoT redundante para realizar o processamento de dados, reduzindo a carga de trabalho no *gateway* IoT primário;
4. **Protótipo**: elaborar protótipos de baixo custo para demonstração da arquitetura com foco em ambientes domésticos.

De acordo com (CHIANG; ZHANG, 2016), há um já antigo movimento de pêndulo entre a centralização do processamento longe dos usuários finais e a distribuição desse processamento mais próximo deles. Há alguns anos, o fluxo tem sido concentrar tudo na nuvem, desde processamento a armazenamento, graças ao advento da *Cloud Computing*. Entretanto esse fluxo começou a mudar de direção com a chegada da Internet das Coisas, que

tem tolerado cada vez menos os atrasos e uma alta latência na comunicação, exigindo mais inteligência próximo aos sensores e atuadores.

A opção de oferecer balanceamento de carga é interessante, mas não traria outros benefícios à rede além da redução da chance de ocorrer congestionamento no *gateway*. Sua aplicação também adicionaria certa complexidade na sincronização de dados entre os *gateways* e, no caso do uso de um balanceador de carga físico, a existência de um ponto único de falha permaneceria.

Por fim, a alternativa de colocar o dispositivo secundário em modo de suspensão de energia apenas reduziria seu gasto, podendo também aumentar o tempo de indisponibilidade caso a mudança de estado fosse mais demorada, como, por exemplo, no caso de o *gateway* IoT redundante permanecesse desligado e precisasse ser manualmente ligado quando necessário.

Por conta dos motivos expostos acima, a proposta deste trabalho de desenvolver um *gateway* IoT redundante com uso de *Fog Computing* é interessante e traz grandes benefícios a uma rede IoT, aumentando sua disponibilidade, eficiência e independência de redes e sistemas externos.

1.3 Organização do Trabalho

Este projeto final de graduação está estruturado em 5 capítulos, além dos anexos:

1 - **Introdução**: descrição dos problemas a serem solucionados e dos objetivos a serem alcançados com a proposta mostrada no presente trabalho.

2 - **Fundamentos Teóricos**: desenvolvimento de conceitos necessários para melhor entendimento dos problemas, das possíveis soluções e da proposta.

3 - **Proposta**: exposição da proposta, com detalhamento dos componentes de *hardware*, dos programas a serem utilizados, do funcionamento e de sua aplicação.

4 - **Resultados**: apresentação dos resultados obtidos após testes e usos em cenários pré-definidos.

5 - **Conclusão**: informações acerca dos resultados obtidos e objetivos atingidos. Breves comentários sobre futuros trabalhos que podem ser realizados após a entrega deste projeto.

I - **Anexos**: guias de instalações, configurações de *softwares* e *scripts* utilizados para montar o protótipo.

Capítulo 2

Fundamentos Teóricos

Este capítulo abordará os principais conceitos, termos e equipamentos que precisam ser descritos e comentados antes de apresentar a proposta e seu funcionamento. O objetivo aqui é explicar porque o *gateway* foi o dispositivo de rede escolhido, qual a relação dos protocolos de comunicação com a Internet das Coisas, o que é redundância e onde *Fog Computing* se encaixa nessa solução.

2.1 Redes de Computadores

2.1.1 Modelo de Referência OSI

Uma rede pode tomar diferentes formas e ser composta por equipamentos distintos, dependendo de seus objetivos, de seu tamanho e das tecnologias disponíveis para se utilizar na ocasião (TANENBAUM; WETHERALL, 2010, p. 54). *Data Centers*, uma rede de uma universidade e uma rede doméstica em um apartamento são exemplos dessa diversidade.

Antes de descrever esses equipamentos e suas funções, é importante examinar um dos modelos de referência utilizados em redes de computadores para guiar o resto do texto. O principal e mais conhecido é o modelo OSI (*Open Systems Interconnection*), desenvolvido pela Organização Internacional para Padronização (*International Standards Organization* - ISO), mostrado na figura 2.1. São sete camadas no total, cada uma possuindo determinadas funções e que, como o próprio nome do modelo diz, servem para orientar como um sistema pode se comunicar com outro:

7. A camada sete ou de aplicação fornecerá funções específicas de cada serviço ou programa, o que motiva todo o envolvimento das outras camadas para criar um canal de comunicação (KUROSE; ROSS, 2013, p. 62). O conjunto de informações a ser enviado ao servidor é chamado de mensagem.

6. A camada seis ou de apresentação serve para adicionar um nível de segurança na conexão e para formatar e converter os dados (TANENBAUM; WETHERALL, 2010, p. 45).

5. Em geral, a camada cinco ou de sessão é utilizada para manter uma sessão ativa em algum site ou programa, garantindo controle de diálogo, sincronização e recuperação

de estado (TANENBAUM; WETHERALL, 2010, p. 44).

4. A camada quatro ou de transporte recebe as mensagens da camada superior e as encapsula em segmentos. Essa camada define basicamente a maneira como os dados serão transmitidos e entregues: a conexão pode ser orientada à conexão, havendo mais garantias de entrega, ou não, visando obter o menor atraso possível. Também é responsável por outras funções, como controle de congestionamento (KUROSE; ROSS, 2013, p. 135).

3. Na camada três ou de rede, os segmentos da camada superior são encapsulados para formar pacotes ou datagramas. São utilizados endereços IP, endereços lógicos dos dispositivos, para realizar o repasse e o roteamento de pacotes (KUROSE; ROSS, 2013, p. 224).

2. Na camada dois ou de enlace, os pacotes entregues pela camada de rede são encapsulados em quadros e baseia-se no endereço MAC, o endereço físico, de cada equipamento. É a camada responsável por detectar e corrigir erros de transmissão entre dois pontos vizinhos e controlar o fluxo de dados a fim de evitar congestionamentos na rede (TANENBAUM; WETHERALL, 2010, p. 194).

1. Na camada um ou física, ocorre a transmissão de dados pelo meio físico, como pulsos elétricos por meio de um cabo, que representam bits de informação (KUROSE; ROSS, 2013, p. 39).

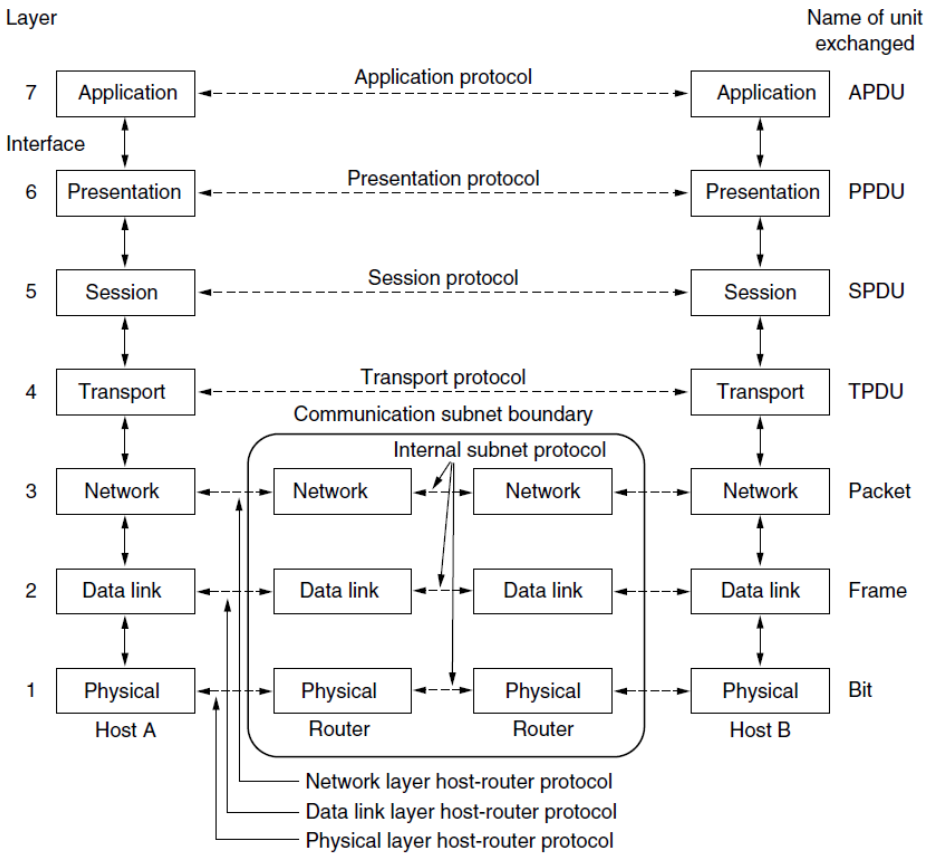

Figura 2.1: O modelo de referência OSI. (TANENBAUM; WETHERALL, 2010, p. 41)

Para que a comunicação ocorra de maneira funcional, é necessário utilizar protocolos, ou seja, um conjunto de regras que "define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento" (KUROSE; ROSS, 2013, p. 7). Além disso, cada protocolo pode ser composto de *hardwares* e/ou *softwares* específicos para ser executado. Quanto mais próximo da primeira camada, a física, mais o protocolo geralmente dependerá de *hardware*, enquanto que quanto mais próximo da sétima camada, de aplicação, mais *software* será utilizado para sua execução (KUROSE; ROSS, 2013, p. 37).

2.1.2 Dispositivos de Redes

A partir da compreensão do modelo OSI, é possível descrever alguns dispositivos de rede com base em quais camadas eles atuam, como pode ser visto na figura 2.2.

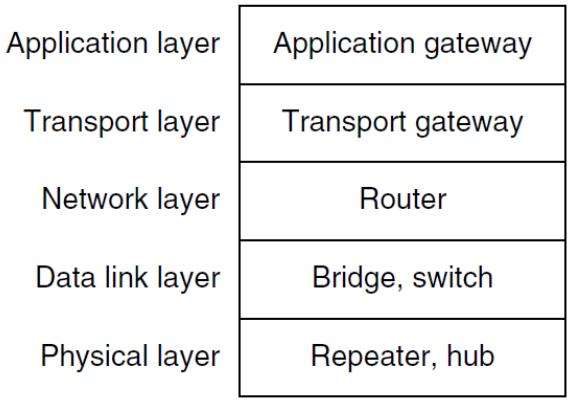

Figura 2.2: Dispositivos de rede por camada. (TANENBAUM; WETHERALL, 2010, p. 340)

Dentre os *hardwares* de rede, os mais simples deles são os **repetidores**. São dispositivos que possuem apenas duas portas e que são utilizados com a intenção de regenerar um sinal recebido de um lado e simplesmente retransmiti-lo pelo outro (TANENBAUM; WETHERALL, 2010, p. 281), atuando apenas a nível de camada física.

Quando é necessário cobrir longas distâncias, o sinal pode perder intensidade e ter ruídos adicionados a ponto de não ser mais possível interpretá-lo no destinatário (FREEMAN, 2005, p. 121). A distância máxima que um sinal pode chegar sem perdas varia de acordo com os tipos de sinal e de meio utilizado para realizar a transmissão.

Um detalhe importante a se notar é que o sinal não é simplesmente amplificado num repetidor, pois isso faria com que o ruído adicionado ao sinal durante o caminho percorrido também fosse amplificado. Se isso acontecesse, o sinal acumularia ruído e chegaria ao destinatário totalmente degradado. O repetidor recebe o sinal com o ruído adicionado durante o percurso, interpreta o sinal e então retransmite-o regenerado. Com isso, o sinal sempre será retransmitido sem ruído e sem perdas ou com a quantidade mínima de erros.

Hubs geralmente possuem a única função de replicar o sinal recebido em uma de suas portas e retransmiti-lo para todas as outras, sem que qualquer outra lógica ou manipulação seja aplicada, o que os torna basicamente um conjunto de repetidores (TANENBAUM; WETHERALL, 2010, p. 341). *Hubs* USB, por exemplo, são bem conhecidos por permitirem

que mais dispositivos USB sejam conectados ao mesmo tempo ao computador do usuário. Assim como os repetidores, *hubs* também atuam apenas na camada física.

Com o uso de *hubs*, existe apenas um canal para que todos os dispositivos conectados utilizem para se comunicar, ou seja, se dois dispositivos tentarem transmitir sinal ao mesmo tempo, ocorrerá uma colisão e as duas transmissões falharão. Pela falta de recursos do *hub*, esse canal de transmissão é bastante ineficiente e já é pouquíssimo usado, principalmente em cenários mais complexos.

Switches, também conhecidos como **comutadores**, são dispositivos que possuem múltiplas portas e que são muito eficientes na retransmissão de quadros da camada de enlace, ou seja, evitam colisões e, ao contrário dos *hubs*, retransmitem sinal apenas para uma porta específica com base no endereço MAC do destinatário, evitando inundar o resto da rede com pacotes desnecessários (KUROSE; ROSS, 2013, p. 352).

Bridges, ou pontes, são semelhantes a *switches*, mas costumam ter menos portas disponíveis (TANENBAUM; WETHERALL, 2010, p. 342). Alguns modelos de *switches* também podem atuar na camada de rede, realizando funções de roteadores. Tais dispositivos são chamados de *Switches Layer-3* ou comutadores de terceira camada.

Roteadores são dispositivos que trabalham na camada de rede e realizam o roteamento de pacotes baseados no endereço IP de destino. Estes dispositivos costumam ficar na borda da rede local, um salto antes da saída para a Internet, por exemplo, ou entre redes distintas, conectando-as.

Para realizar esse roteamento, é criada uma tabela que possui informações sobre cada rede e por qual porta elas estão chegando ao roteador. Desse modo, quando um pacote chega ao roteador para ser enviado a outra rede, ele consultará essa tabela e realizará o repasse de acordo com as informações gravadas (KUROSE; ROSS, 2013, p. 235). A figura 2.3 mostra a diferença de processamento de pacotes entre *switches*, roteadores e dispositivos finais.

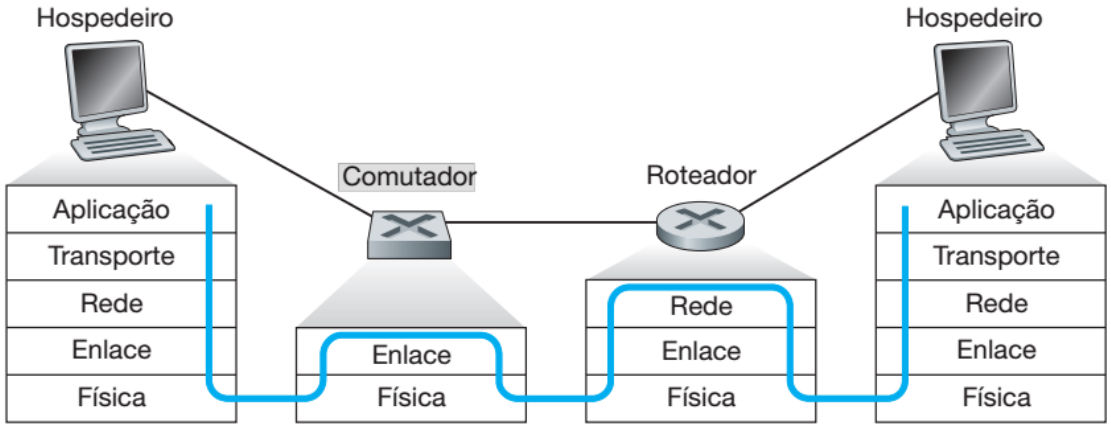

Figura 2.3: Processamento de pacotes em *switches*, roteadores e hospedeiros. (KUROSE; ROSS, 2013, p. 356)

2.1.3 *Gateways*

Gateway é um termo abrangente que pode ser empregado em dispositivos que fazem a tradução de um protocolo para o outro, permitindo a comunicação entre hospedeiros em redes distintas (TANENBAUM; WETHERALL, 2010, p. 342). Por poder se comunicar utilizando diferentes protocolos, é um equipamento capaz de atuar em diversas camadas, por vezes assumindo o nome onde está sendo aplicado, como *gateway* de aplicação ou *gateway* de transporte.

Como o foco da solução a ser apresentada está em redes IoT, onde os dispositivos precisam se comunicar utilizando diferentes protocolos que nem sempre utilizam IP, como *Bluetooth* e *Zigbee*, então há a necessidade de realizar traduções de protocolos de múltiplas camadas. Por isso, o responsável por essa função é chamado de ***Gateway* IoT**.

De acordo com (ZHU et al., 2010), um *gateway* IoT tem as seguintes características:

1. Possuir suporte a uma variedade de protocolos para possibilitar a comunicação entre uma rede IoT e redes tradicionais.
2. Permitir a gerência de vários dispositivos IoT, para identificar, controlar e analisá-los; e do seu próprio sistema, a fim de viabilizar configurações, atualizações e manutenções, por exemplo.

Além disso, ainda com base em (ZHU et al., 2010), as funções básicas de um *gateway* IoT são:

1. Repasse de dados, ou seja, recolher dados e realizar o *upload* deles para uma aplicação na rede local ou para a nuvem.
2. Conversão de protocolo, para permitir que dados que venham de conexões que não utilizam TCP/IP cheguem à Internet, por exemplo.
3. Controle e gerenciamento da rede IoT, ou seja, permitir que atuadores sejam acionados, a partir ou através do *gateway*.

2.2 Internet das Coisas

O conceito de *Gateway* IoT foi abordado, mas ainda sem explicar o que é de fato uma rede IoT. A **Internet das Coisas**, ou simplesmente IoT (*Internet of Things*), é um termo cunhado para se referir ao uso da rede por não mais apenas pessoas e sim também por objetos, máquinas e sistemas de várias áreas da sociedade (AL-FUQAHA et al., 2015).

Com a evolução da tecnologia utilizada no campo da eletrônica e de telecomunicações, dispositivos passaram a se comunicar com pessoas e outras máquinas, enviando dados e requisições diversos pela rede de maneira autônoma. Como visto no capítulo 1, a quantidade de dispositivos IoT e de dados gerados por eles tende a crescer exponencialmente nos próximos anos.

A palavra *smart* se tornou popular na última década para identificar dispositivos com conexão com a Internet e/ou com poder de processamento. Quando eles não possuem esses recursos, são considerados dispositivos burros ou *dumb*, o que significa na maioria dos casos que eles são apenas analógicos ou mecânicos. Os telefones tornaram-se smartphones, os relógios agora são smartwatches e as televisões mais simples perderam espaço para as *smart* TVs. Esses produtos ganharam cada vez mais sensores e poder de processamento para facilitar a vida do usuário e oferecer mais serviços e conteúdo.

Redes IoT podem possuir, por exemplo, redes de sensores sem fio, ou WSNs (*Wireless Sensor Networks*), que são redes compostas por apenas dispositivos sensores e atuadores que se comunicam entre si ou com um *gateway* através de comunicação sem fio, como Wi-Fi, *Bluetooth* ou *Zigbee* (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

2.2.1 Sensores e Atuadores

Os dispositivos IoT são basicamente divididos em dois grupos: **sensores** e **atuadores**. São dispositivos geralmente com pouco poder de processamento e armazenamento, mas com algum tipo de interface de comunicação e às vezes são alimentados por baterias (HUNKELER; TRUONG; STANFORD-CLARK, 2008). Podem ser denominados como dispositivos SA (sensores e atuadores).

O primeiro grupo capta dados do ambiente em que está inserido, como sensores de temperatura, de umidade, de luminosidade, sensores biométricos, microfones e câmeras. Os sensores convertem um tipo de energia em outra, geralmente utilizando tensões e correntes para realizar a medição, como transformar a tensão nos terminais de um LED foto-sensível em nível de luminosidade de uma sala.

O segundo grupo é responsável por realizar uma ação, fazendo basicamente o caminho inverso da conversão de energia de um sensor: uma tensão ou corrente é gerada e então, a partir dela, uma ação física é realizada, como o acionamento de um motor ou de um relé.

Na Internet das Coisas, dispositivos dos dois tipos geralmente estão integrados e trabalham em conjunto, como sensores de presença e relés que acendem lâmpadas, por exemplo. Apesar de o fluxo de dados ser geralmente de saída dos sensores e de entrada nos atuadores, ambos podem receber dados referentes a configuração e gerenciamento (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

2.3 *Middleware* IoT

A maioria dos dispositivos IoT não possui grande capacidade de armazenamento e processamento. Por isso, os dados gerados por esses dispositivos são enviados para outro lugar. Para que tais dados sejam coletados e armazenados para futura análise, algumas aplicações foram desenvolvidas, as quais são chamadas de ***middleware*** (AAZAM; HUH, 2014).

Um *middleware* permite a interação entre dados e aplicações ou entre aplicações. Uma aplicação pode acabar fazendo parte do *middleware*, caso se torne essencial para que a arquitetura montada funcione. O *middleware* também controla e monitora o estado dos dispositivos, podendo configurar e gerenciar sensores e atuadores ou permitir que outra aplicação faça isso.

Um *middleware* costuma estar na nuvem ou em um dispositivo na borda da rede, como um *gateway*, de onde consegue uma posição mais centralizada e pode ter acesso a mais dispositivos e dados.

Por exemplo, o ThingSpeak (THE MATHWORKS, INC, 2019) é um *middleware* popular que foi desenvolvido para armazenar dados de dispositivos IoT na nuvem. Os dados podem ser visualizados diretamente pela aplicação e tratados posteriormente com o auxílio do MATLAB.

2.4 Protocolos de Comunicação

Após a revisão da parte mais física de redes IoT, composta de dispositivos SA e dispositivos de rede, é válido seguir para os protocolos de comunicação que são utilizados nesses ambientes.

Existem inúmeros protocolos de comunicação, cada um com suas próprias características e usos específicos. Os mais relevantes serão abordados nas subseções seguintes de maneira breve, seguindo novamente as camadas do modelo OSI. Para os próximos exemplos, suponha-se que existem 2 dispositivos, A e B, que desejam se comunicar. A será o cliente se ele for o dispositivo que está iniciando a comunicação, enquanto B será denominado como servidor, respondendo requisições do cliente.

2.4.1 *Ethernet*

Definida pelo padrão IEEE 802.3 e com várias versões desenvolvidas, a **Ethernet** é uma tecnologia utilizada para realizar transmissão de dados através de cabos, tanto por par trançado quanto por fibra óptica (KUROSE; ROSS, 2013, p. 351). É um protocolo de camada física e de enlace.

Foi criada na década de 1970 e se popularizou para o uso em redes locais (LAN). Até hoje, a *Ethernet* é a tecnologia que predomina no mercado. Ela possui suporte a várias velocidades diferentes, chegando a números incríveis em ambientes de *Data Center*, como 40GbE e 100GbE (40 Gigabits/segundo e 100 Gigabits/segundo, respectivamente).

Por ser uma tecnologia que exige conexão cabeada, a mobilidade acaba sendo mínima, já que o dispositivo conectado precisa se manter ligado a um ponto de rede. Em ambientes IoT, uma conexão *Ethernet* pode ser útil para dispositivos que não serão mudados de lugar e que precisam de uma conexão mais confiável e com mais banda disponível.

2.4.2 Wi-Fi

O **Wi-Fi** é uma tecnologia baseada no padrão IEEE 802.11. É a tecnologia mais popular no mundo para permitir a comunicação sem fio entre dispositivos (WI-FI ALLIANCE, 2018). Vários padrões foram desenvolvidos a partir do original, mas em geral as faixas de frequência utilizadas são a 2.4 GHz e a 5.8 GHz (KUROSE; ROSS, 2013, p. 339). Assim como a Ethernet, o Wi-Fi é um protocolo que atua nas camadas física e de enlace.

Uma conexão sem fio permite maior mobilidade aos dispositivos, mas pode sofrer mais

com interferência de outras fontes, como micro-ondas de cozinha, por exemplo, diminuindo a taxa de transmissão de dados ou impedindo completamente a conexão.

2.4.3 TCP e UDP

Ambos protocolos são utilizados a nível de camada de transporte do modelo OSI. São muito importantes e bastante utilizados na comunicação pela Internet.

O *Transmission Control Protocol*, ou **TCP**, é um protocolo orientado à conexão (KUROSE; ROSS, 2013, p. 169). Ele provê garantias na entrega de dados, como a ordenação e a integridade dos mesmos. TCP suporta apenas conexão *unicast*, ou seja, apenas a transmissão entre dois dispositivos por vez. TCP é bastante utilizado em aplicações em que a integridade dos dados e sua entrega são mais importantes do que o atraso que os pacotes podem sofrer até chegar ao servidor, como o envio de mensagens de texto, fotos, e-mails e transferência de arquivos. Além disso, a conexão via TCP é *full-duplex*, ou seja, o dispositivo A pode enviar dados para B ao mesmo tempo em que B envia dados para A sem que haja colisão de transmissões.

Para estabelecer a conexão entre dois dispositivos, ocorre o que é chamado de *three-way handshake*, ou seja, a troca de três mensagens SYN, SYN-ACK e ACK para iniciar a comunicação. Para a ordenação, cada pacote enviado possui um número de sequência, então o destinatário pode reordenar os dados recebidos mesmo que cheguem fora de ordem. Em caso de perda de dados durante a transmissão, o TCP pode retransmiti-los até que sejam de fato entregues e que tenham a entrega confirmada.

Há também controle de fluxo e de congestionamento, ajustando a taxa com que um dispositivo envia dados, chamada de janela de congestionamento, para que a transmissão de dados fique sempre próxima do limite que a conexão suporta, sem que isso cause congestionamento de tráfego na rede e retransmissões.

Por fim, o TCP também oferece verificação de integridade dos dados recebidos, para detectar erros inseridos durante a transmissão. Se algum erro for detectado no servidor, ele não enviará o ACK referente àquele pacote e uma retransmissão será realizada.

O *User Datagram Protocol*, ou **UDP**, é um protocolo bem mais simples, com menos recursos que o TCP e não é orientado à conexão, ou seja, não há confirmação da entrega dos dados (KUROSE; ROSS, 2013, p. 145). O UDP é muito utilizado em aplicações em tempo real, ou seja, quando os dados são sensíveis ao tempo e que perda de parte deles é aceitável, como, por exemplo, em ligações e em transmissões de vídeos ao vivo. Nesses casos, os dados recebidos em atraso não serão mais úteis, pois a sequência onde eles estavam inseridos já foi consumida pelo usuário.

Por ter como característica não necessitar realizar a sincronização da conexão entre dois dispositivos antes de iniciar o envio de dados, o UDP pode ser utilizado tanto em transmissões *unicast* quanto em *multicast* ou *broadcast*.

2.4.4 HTTP

O *Hypertext Transfer Protocol*, ou **HTTP**, é um protocolo da camada de aplicação para exibir informações na Internet (IETF, 1999). O HTTP se baseia na arquitetura servidor-

cliente, em que o protocolo é executado nos dois e mensagens de requisição e resposta são trocadas entre eles para estabelecer a conexão e realizar a transmissão de dados (KUROSE; ROSS, 2013, p. 72).

2.4.5 MQTT

O **MQTT**, ou *MQ Telemetry Transport*, foi criado explicitamente com o objetivo que ser um protocolo para Internet das Coisas, mais especificamente para comunicações máquina a máquina, ou M2M (MQTT.ORG, 2018). É um protocolo diferente dos usuais baseados na arquitetura cliente-servidor. A arquitetura do MQTT é composta de dispositivos que publicam mensagem, os *publishers*, de dispositivos que se inscrevem para receber mensagens, os *subscribers*, e de um servidor central chamado *broker*, que recebe as mensagens dos *publishers* e as repassa para os *subscribers* (THE HIVEMQ TEAM, 2015a), como mostrado na figura 2.4.

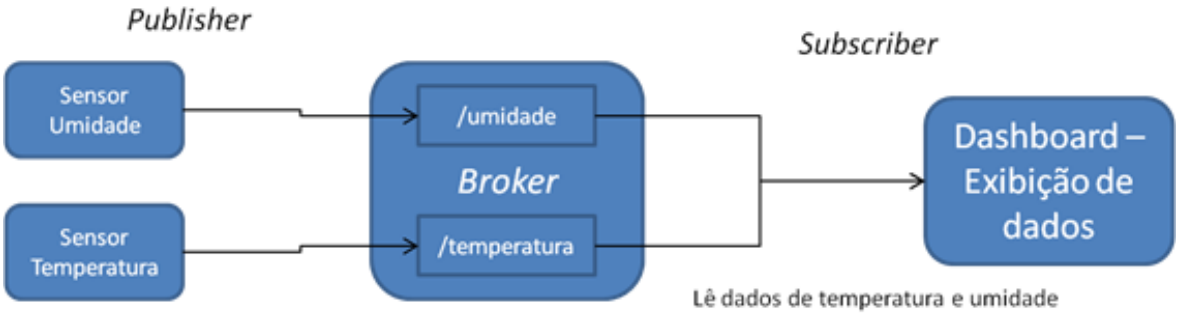

Figura 2.4: Funcionamento do protocolo MQTT. (FIWARELAB, 2016)

As mensagens possuem hierarquia e são estruturadas em tópicos, como se fossem diretórios em um sistema operacional. Os *publishers* enviam dados para o *broker* em um certo tópico e os *subscribers* se inscrevem no *broker* para receber mensagem de tópicos específicos. Se há, por exemplo, uma lâmpada inteligente na cozinha de uma casa e deseja-se receber no cliente do smartphone as mensagens que essa lâmpada está enviando, basta se inscrever no *broker* para receber mensagens do tópico "/casa/cozinha/lâmpada", que o *broker* repassará as mensagens da lâmpada para o cliente do smartphone. Caso deseja-se receber dados de todos os dispositivos da cozinha, basta se inscrever no tópico "/casa/cozinha", e assim por diante (THE HIVEMQ TEAM, 2015b).

2.4.6 *Bluetooth* e *Zigbee*

Bluetooth é um protocolo utilizado para realizar comunicação sem fio entre dispositivos a curta ou média distância na faixa de frequência de 2.4 GHz, a mesma do Wi-Fi, porém está num grupo separado, o padrão IEEE 802.15.1 (KUROSE; ROSS, 2013, p. 402). O *Bluetooth* é um conjunto de protocolos que não se encaixa em nenhum modelo de referência (TANENBAUM; WETHERALL, 2010, p. 322).

O *Bluetooth Low Energy* (BLE) foi criado a partir da versão 4.0 do *Bluetooth* com o intuito de manter conexões de curto alcance, mas com um consumo de energia para transmissão de sinal muito menor, o que possibilitou sua popularização em *smartphones* e incen-

tivou a criação de novos *gadgets* que ficam conectados praticamente todo o tempo com o *smartphone*, como *smartbands* e *smartwatches* (DAVIDSON et al., 2019).

O ***Zigbee*** é outro padrão de comunicação sem fio, definido no IEEE 802.15.4 (ZIGBEE ALLIANCE, 2014). Possui características bem parecidas com o BLE, como o uso da faixa de 2.4 GHz de frequência e possuir alcance e consumo de energia baixos.

O *Zigbee* é um conjunto de protocolos que atuam em todas as camadas superiores, com o padrão 802.15.4 atuando nas camadas física e de enlace. É uma tecnologia voltada para comunicações mais simples, com pequeno volume de dados para ser enviado, como interruptores e dispositivos de segurança, por isso é mais popular em automações residenciais. É uma alternativa ao *Bluetooth* e ao Wi-Fi (CHARARA, 2018) (KUROSE; ROSS, 2013, p. 403).

2.5 *Ubiquitous Computing*

Com dispositivos finais conseguindo meios para se comunicar com a rede e ganhando poder de processamento, a Internet das Coisas foi ganhando forças e se tornando cada vez mais presente na sociedade. A **computação ubíqua ou pervasiva** é um conceito relacionado a essa presença da computação de maneira intrínseca e onipresente na vida das pessoas.

Antigamente, a computação estava limitada a certos locais ou dispositivos, como video games, computadores de mesa e caixas eletrônicos, mas hoje em dia ela está presente em praticamente qualquer lugar ou coisa, como carros, relógios e até mesmo roupas (SAWH, 2018).

(WEISER, 1991) disse que "a tecnologia necessária para a computação ubíqua é composta por três partes: computadores baratos de baixa potência que incluem telas igualmente conveniente, uma rede que una-os e sistemas de software que implementem aplicações ubíquas". Aos poucos, tecnologias se desenvolveram o suficiente para fazer parte da rotina de muita gente, apesar de não ser facilmente percebida. Por exemplo, relógios são comuns há muitos anos, mas recentemente surgiram os smartwatches, cheios de sensores e que analisam o ambiente e o usuário constantemente, provendo mais informações do que um simples relógio analógico. Carros que dirigem sozinhos, sem interferência humana, são outro exemplo de computação ubíqua.

O impacto que a aplicação de tais tecnologias possuem na sociedade varia bastante, mas em geral todo o processamento por trás de ações realizadas por dispositivos está basicamente oculto para os usuários. Isso pode ser percebido ao saber que já existem "assistentes virtuais"que conversam com pessoas reais para realizar a compra de algo ou a marcação de algum compromisso, de modo que a pessoa não percebe que está falando com uma máquina (WELCH, 2018).

2.6 *Cloud Computing*

Outros conceitos relacionados à computação surgiram nos anos seguintes. Após o surgimento da virtualização, empresas especializadas em *Data Centers* começaram a popularizar e baratear seus serviços para outras empresas e pessoas através da Internet. Produtos como a

Amazon Web Services (AWS), a Microsoft Azure e a Google Cloud Platform (GCP) surgiram e ficaram disponíveis para quem desejasse obter acesso a poder de processamento e de armazenamento remoto. Uma drástica mudança iniciou-se: a maioria das empresas percebeu então que não é necessário mais manter equipamentos *on-premise*, ou seja, possuir servidores, *storages*, *racks* e toda uma infraestrutura física de um mini *Data Center* em seu próprio ambiente, além de cuidar de toda a manutenção envolvida (MAHESA, 2018).

Esses serviços ficaram conhecidos como ***Cloud Computing***, ou computação em nuvem, e em geral são utilizados para hospedar sites e aplicações na Internet. Por conta da virtualização, é possível aumentar e diminuir os recursos (processamento e armazenamento) alocados para uma certa aplicação de acordo com a situação de cada momento, tendo como benefícios redução de investimentos, maior flexibilidade e alta disponibilidade (ARCITURA EDUCATION INC, 2019).

Por exemplo, se um site passar a receber muito mais acessos de uma hora para a outra, mais recursos podem ser alocados para mantê-lo online e servindo o tráfego extra, algo que não seria possível caso o site estivesse sendo hospedado em um simples computador no limite de seu processamento. Numa situação inversa, quando não há muitos acessos em um determinado horário, os recursos disponíveis para o site podem ser reduzidos para diminuir os custos tanto para o cliente, que não estará utilizando recursos desnecessários da nuvem, quanto para a empresa que mantém a nuvem, que gastará menos com energia elétrica ou poderá oferecer aquele recurso agora disponível para outro cliente.

Atualmente, **Cloud Computing** está intimamente ligada ao conceito de ***Big Data***, já que é comum levar grande volume de dados até a nuvem e processá-los por lá. Por vezes, tais dados são gerados por sensores, logo, IoT, Big Data e Cloud Computing são conceitos que costumam se entrelaçar constantemente (AL-FUQAHA et al., 2015).

Com a evolução da Internet das Coisas, as desvantagens da **Cloud Computing** tornaram-se mais perceptíveis. Considerando o simples cenário de utilizar um celular para ligar uma lâmpada de casa através de uma aplicação hospedada na nuvem, o usuário enviará o comando de ligar a lâmpada para a Internet, até chegar na nuvem e enfim a aplicação receberá o comando. Ela então enviará uma requisição até o atuador local que controla o estado da lâmpada, finalmente acendendo-a. Todo esse processo pode levar até alguns segundos, dependendo de onde a aplicação está hospedada e isso considerando que no momento há conexão com a Internet.

2.7 *Fog Computing*

Para situações em que dados são sensível à latência, o ideal seria estender o poder de processamento e armazenamento da nuvem para a borda da rede. Esse conceito ficou conhecido como ***Fog Computing***, termo cunhado pela Cisco (LINTHICUM, 2019). A ideia consiste em disponibilizar esses recursos em um dispositivo próximo ou dentro da rede local, como um *gateway*, permitindo que dados sejam pré-processados e ações sejam tomadas de maneira mais rápida.

Voltando ao cenário anterior, em vez de a requisição ir até à nuvem, ela já pode ser processada em um *gateway* local, permitindo que a lâmpada seja acesa em questão de milissegundos. Além disso, mesmo que não haja conexão com a Internet, a requisição será atendida, já que a requisição não precisa sequer sair da rede local.

Dados que são sensíveis ao tempo e que permitem a atuação de algum dispositivo são processados no *gateway* na borda da rede (CHIANG; ZHANG, 2016). Dados que não são sensíveis ao tempo, ou seja, que não exigem que alguma ação seja tomada assim que são recebidos, serão úteis em grandes quantidades na nuvem, facilitando, por exemplo, a manutenção preventiva dos equipamentos que estão gerando esses dados.

IoT Challenges	How Fog Can Help
Latency Constraints	Fog, performing data analytics, control, and other time-sensitive tasks close to end users, is the ideal and often the only option to meet the stringent timing requirements of many IoT systems.
Network Bandwidth Constraints	Fog enables hierarchical data processing along the Cloud-to-Things continuum, allowing processing to be performed where it can balance between application requirements and available networking and computing resources. This also reduces the amount of data that needs to be sent to the Cloud.
Resource-Constrained Devices	Fog can carry out resource-intensive tasks on behalf of resource-constrained devices when such tasks cannot be moved to the Cloud due to any reason, hence reducing these devices' complexity, lifecycle costs, and energy consumption.
Uninterrupted Services with Intermittent Connectivity to the Cloud	A local Fog system can operate autonomously to ensure non-interrupted services even when it has intermittent network connectivity to the Cloud.
New IoT Security Challenges	A Fog system can, for example, 1) act as the proxies for resource-constrained devices to help manage and update the security credentials and software on these devices, 2) perform a wide range of security functions, such as malware scanning, for the resource-constrained devices to compensate the limited security functionality on these devices, 3) monitor the security status of nearby devices, and 4) take advantage of local information and context to detect threats on a timely manner.

Figura 2.5: *Fog Computing* provê meios efetivos de enfrentar desafios IoT (CHIANG; ZHANG, 2016)

O *gateway* na borda da rede também poderá reduzir o consumo de banda, já que poderá pré-processar e enviar um resumo dos dados recebidos pelos sensores à nuvem.

2.8 *Mist Computing*

Por fim, temos a ***Mist Computing***. Esses três termos (*cloud*, *fog* e *mist*) foram utilizados por conta da seguinte analogia: a Internet é o céu enquanto que computadores, smartphones, sensores e atuadores são o solo. A *Cloud Computing* (nuvem) está localizada na Internet (no céu), a *Fog Computing* (neblina) está entre a Internet e os dispositivos (entre o céu e o solo) e a *Mist Computing* (névoa) está bem próximo ao solo (aos dispositivos).

Assim como a *Fog Computing*, a *Mist Computing* seria a extensão do processamento e armazenamento até bem perto dos sensores e atuadores, de tal modo que eles teriam certa independência tanto da nuvem quanto da borda da rede. Podemos considerar que smartphones fazem parte da Mist Computing, já que eles possuem sensores e poder de processamento e armazenamento (MAIER; SAME, 2018). Um Arduino com um sensor de temperatura que envia dados para a nuvem ou para o *gateway* já possui um certo poder de processamento em si e compõe a *mist*.

É importante ressaltar que esses três tipos de computação são complementares (MAHESA, 2018), mas podem ser independentes. E quanto mais próximo dos dispositivos finais, mais descentralizado é o sistema. A figura 2.6 mostra a arquitetura hierárquica dos três conceitos de computação comentados nessa seção, indicando diferenças que possuem, como quantidade de dados gerados e latência.

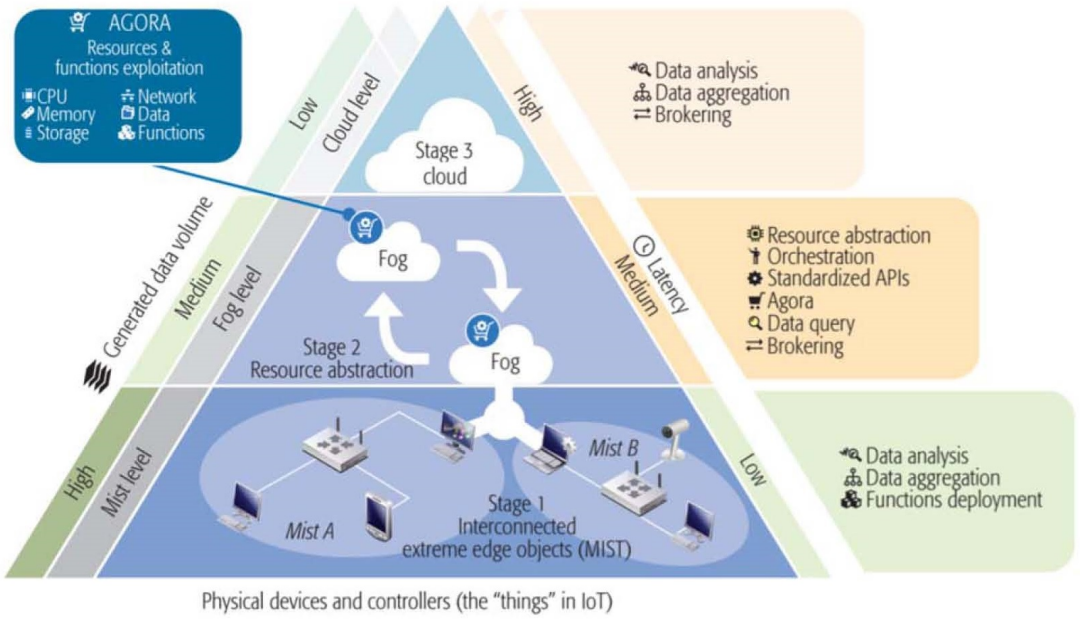

Figura 2.6: Arquitetura hierárquica baseada em *Mist*, *Fog* e *Cloud Computing*. (MARKAKIS et al., 2017)

Cada conceito possui suas vantagens e desvantagens e eles devem ser utilizados em conjunto para maximizar seus potenciais. Ter processamento perto do sensor permite uma rápida resposta e reduz a utilização de banda da rede, mas encarece o armazenamento e dificulta a realização da análise histórica de seus dados.

Há também na indústria e na academia o termo *Edge Computing*, mas seu significado é mais abrangente e varia bastante entre pessoas e empresas. Este conceito trata de computação

na borda da rede, assim como *Fog Computing*, mas para evitar ambiguidades, essa expressão não será utilizada neste trabalho.

Nas últimas seções, os termos disponibilidade e manutenção foram citados e, por terem conexão com redundância, serão abordados a seguir.

2.9 Alta Disponibilidade

Alguns sistemas, serviços ou dispositivos possuem um alto nível de criticidade, a ponto de gerar problemas ou prejuízos a pessoas ou empresas caso fiquem indisponíveis, mesmo que seja por apenas alguns segundos. Tais recursos necessitam de **alta disponibilidade**, ou seja, que possuam o mínimo de tempo possível fora do ar ou desativados.

Se um sistema possui disponibilidade de 90%, então ele ficará 10% do tempo indisponível. Isso significa que, durante os 365 dias do ano, o sistema permanecerá fora do ar por volta de 36,5 dias.

Para estimar esse período de disponibilidade, levam-se em conta dois indicadores: o tempo médio entre falhas, ou **MTBF** (*mean time between failures*), e o tempo médio para reparo, ou **MTTR** (*mean time to repair*) (OPSERVICES, 2015).

O MTBF é calculado da seguinte maneira:

MTBF = (tempo total de disponibilidade – tempo total de indisponibilidade)/(número total de falhas)

Já o MTTR é calculado utilizando a fórmula a seguir:

MTTR = (tempo total gasto realizando reparos)/(número total de falhas)

Por fim, a disponibilidade em porcentagem é calculada utilizando os dois indicadores:

Disponibilidade = MTBF/(MTBF + MTTR)

A intenção é aumentar o MTBF, ou seja, fazer com que menos falhas ocorram e que elas demorem para acontecer, e reduzir o MTTR, ou seja, diminuir o tempo de reparo. Para realizar essas melhorias, dois conceitos podem ser aplicados: manutenções e redundâncias.

2.9.1 Manutenções

Existem 3 tipos de manutenções: **manutenções corretivas**, quando ocorre uma falha e só então uma ação é tomada para que o problema seja resolvido e faça com que o sistema ou dispositivo volte a funcionar; **manutenções preventivas**, que são atuações periódicas com a intenção de prevenir a ocorrência de falhas já conhecidas; e **manutenções preditivas**, que se baseiam no constante monitoramento do sistema ou dispositivo para identificar que uma falha pode acontecer num futuro próximo (SWANSON, 2001).

Manutenções corretivas costumam ser caras, emergenciais e podem causar grande indisponibilidade do recurso, impactando principalmente o MTTR. As preventivas e preditivas aumentam bastante a disponibilidade, evitando falhas e aumentando o MTBF. Nem toda característica pode ser monitorada ou vale a pena monitorar, então a combinação desses dois tipos de manutenção faz parte da rotina de vários sistemas.

2.9.2 Pontos únicos de falha e Redundância

Quando uma falha ocorre num equipamento e ele fica indisponível, é possível que toda a cadeia de processos na qual ele está inserido também pare. Nesse caso, esse equipamento é chamado de **ponto único de falha**. Para aumentar a disponibilidade dessa cadeia de processos, é necessário adicionar **redundância** a todos seus pontos únicos de falha (ORACLE CORPORATION, 2010).

Com a redundância, se um equipamento falhar, outro pode assumir suas funções para que o sistema continue funcionando. Esse equipamento adicional pode ser igual ou equivalente ao que falhou ou pode simplesmente assumir as funções essenciais que ficaram indisponíveis, para manter o sistema no ar até que a falha seja corrigida e tudo seja normalizado.

A redundância pode ser composta de um ou mais dispositivos ou sistemas: quanto mais recursos redundantes houver, maior será a garantia de disponibilidade. A comutação do recurso titular para o de *backup* também influencia: se a comutação for automática, a reação em caso de falha pode ser imediata e minimizar o tempo entre falhas.

2.10 Balanceamento de Carga

Balanceamento de carga é a distribuição de requisições, sessões ou processos entre um conjunto de dispositivos, geralmente servidores, para evitar que um dispositivo fique sobrecarregado. Como há mais de um servidor respondendo requisições, as respostas são feitas de maneira mais rápida. Isso também implica em alta disponibilidade, já que a falha de um servidor pode manter o sistema funcionando, mesmo que com atrasos nas respostas (NGINX, 2019).

Esse conjunto de servidores pode ser chamado de *cluster*, *pool* ou *farm*. O balanceador de carga fica posicionado na entrada desse *cluster*, realizando a distribuição para os servidores. Entretanto, se ele for único no sistema, pode se tornar um ponto único de falha.

O balanceamento pode ser realizado de diversas formas, como, por exemplo: distribuir um número fixo de requisições para cada servidor sequencialmente, entregar requisições para o servidor menos ocupado ou definir servidores para responder requisições de certos grupos de clientes, divididos por endereço de IP, rede, região, etc.

O balanceador de carga pode ser físico, como um equipamento dedicado a realizar essa função, ou virtual, como uma máquina virtual (*virtual machine*, ou VM) ou um *software* em execução.

2.11 Protocolos de Redundância de *Gateway*

Como os conceitos de redundância e alta disponibilidade já foram abordados, é possível comentar sobre os Protocolos de Redundância de Primeiro Salto (ou *First Hop Redundancy Protocol* - FHRP). São protocolos que permitem criar redundância no dispositivo localizado na borda da rede local que é o destino do primeiro salto dos hospedeiros, ou seja, o *gateway* padrão dos dispositivos da rede local. Alguns desses procotolos serão descritos nas subseções seguintes.

Em geral, protocolos FHRP são bem parecidos e sempre criam um endereço IP virtual para um *cluster* de *gateways*. Em vez de os dispositivos se conectarem diretamente aos *gateways*, eles se conectarão a esse endereço IP virtual. Um dos *gateways* do conjunto assumirá então o IP virtual e passará a responder as requisições normalmente. A partir desse momento, ele será denominado de *gateway* ativo, principal ou mestre, dependendo da nomenclatura do protocolo que está sendo utilizado.

Os outros componentes do *cluster* ficarão em estado de espera e monitorando o funcionamento do *gateway* ativo. Esses serão denominados *gateways* secundários, em espera ou de *backup*. Se os componentes em espera perceberem que o *gateway* ativo não é mais encontrado na rede, um deles assumirá o IP virtual, o qual passará a responder as requisições como novo *gateway* ativo, mantendo assim a rede em funcionamento mesmo que o *gateway* anterior tenha entrado em estado de falha.

Os componentes do *cluster* mantêm informação sobre o estado dos outros utilizando um esquema chamado de *heartbeating*, ou seja, um dispositivo envia um pacote de notificação para o outro em intervalos fixos para sinalizar que ainda está ativo na rede. É mais comum o componente ativo enviar um sinal *multicast* para os outros componentes do *cluster*, evitando assim inundar a rede com pacotes de sinalização.

Quanto menor o intervalo, mais pacotes de sinalização serão enviados em um certo período de tempo, então mais rápido será para um *gateway* em espera perceber que houve falha no *gateway* primário, diminuindo assim o tempo de indisponibilidade da rede. Por outro lado, mais pacotes de sinalizações estariam trafegando na rede, podendo causar congestionamentos de dados.

2.11.1 *Hot Standby Router Protocol* (HSRP)

É um protocolo proprietário da Cisco utilizado para prover alta disponibilidade a alguns produtos da empresa. Quando o HSRP é configurado em dois ou mais dispositivos em uma mesma rede, um *gateway* virtual com endereços MAC e IP é criado para representar esse grupo de *gateways* reais. Após um processo de seleção baseada em critérios como prioridade e endereço IP do dispositivo, os endereços virtuais serão utilizados por aquele que for escolhido como ativo (CISCO, 2017).

Um segundo *gateway* ficará em espera (*standby*) enquanto os outros do *cluster* permanecerão inativos. O *gateway* em espera estará pronto para assumir os endereços virtuais caso o *gateway* ativo anteriormente selecionado falhe. A comunicação para detectar falhas e para definir papéis de estado ativo ou em espera é feita através de pacotes transmitidos por *multicast*.

O HSRP não possui recursos visando garantir segurança ou balanceamento de carga entre os componentes do *cluster*. Por ser proprietário da Cisco, sua utilização em soluções caseiras ou em trabalhos acadêmicos é dificultado.

2.11.2 *Virtual Router Redundancy Protocol* (VRRP)

Considerando sua terceira versão, descrita na RFC 5798 (IETF, 2010a), é um FHRP popular por ser um *software open source* e ter praticamente as mesmas características e funcionamento que o HSRP. Por conta dessas similaridades, houve disputas judiciais entre o

IETF (*Internet Engineering Task Force*), a Cisco, a IBM e outras empresas quando o VRRP foi considerado um padrão aberto.

O VRRP representa um *cluster* de dispositivos através de um *gateway* virtual com endereços MAC e IP. Para o VRRP, o *gateway* ativo é denominado mestre e os outros são chamados de *gateways* de *backup* ou de apoio. Cada *gateway* possui um número de prioridade entre 1 e 254, onde aquele que possui o maior valor é eleito como mestre, passando a ter então prioridade igual a 255. Caso haja *gateways* com a mesma prioridade máxima, aquele com o maior IP será então eleito como mestre.

Os pacotes de sinalização são chamados de anúncios. No VRRP, apenas o *gateway* mestre envia anúncios para os *gateways* de *backup* por *multicast*, informando seu estado atual e outros dados. Se o mestre parar de anunciar seu estado pelo período de três anúncios consecutivos, o *gateway* de *backup* com a maior prioridade assumirá o papel de mestre.

Há uma configuração chamada de preempção que, quando ativa, faz com que o dispositivo disponível com a maior prioridade sempre assuma o papel de mestre. Ou seja, se um *gateway* com prioridade 244 for o mestre, entrar em estado de falha, for substituído por outro com prioridade 100 e, depois algum tempo, voltar a ficar disponível após a correção da falha, esse *gateway* com prioridade 244 voltará a ser mestre assim que possível, fazendo aquele com prioridade 100 voltar ao estado de *backup*.

A ativação desse recurso pode reduzir a disponibilidade desse ponto da rede, já que haverá mais trocas de mestres. Por isso, em casos críticos, é recomendado que a preempção permaneça inativa, para evitar indisponibilidades, principalmente em casos onde o *gateway* com maior prioridade sofrer de falhas intermitentes.

Um recurso adicional do VRRPv3 é a possibilidade de oferecer balanceamento de carga entre o *gateway* mestre e os de *backup*. A segurança também é reforçada contra ataques de fora da rede local através da verificação do TTL (time to live, número de saltos que um pacote deu até chegar em algum nó da rede) de pacotes que buscam influenciar na estrutura do *gateway* virtual. Caso um pacote com mais de um salto (TTL < 255) tenha sido recebido pelos componentes do *cluster*, esse pacote é descartado, porque ele estará vindo de fora da rede local. Entretanto, recursos como autenticação ou proteção contra ataques vindos de dentro da rede local não foram implementados ou foram retirados por não garantirem proteção suficiente.

2.11.3 *Gateway Load Balancing Protocol* (GLBP)

GLBP é mais um protocolo proprietário da Cisco. Para compensar a falta de alguns recursos no HSRP, o GLBP pode prover nativamente balanceamento de carga entre os componentes do *cluster* de *gateways* e autenticação para melhorar a segurança contra tentativas de ataque (CISCO, 2019).

O HSRP e o VRRP podem ser utilizados para garantir balanceamento de carga manualmente. Para isso, um novo *cluster* virtual incluindo todos os *gateways* deve ser criado para cada *gateway* que se deseja utilizar na distribuição de carga. Em cada *cluster*, um *gateway* diferente deve ser definido como ativo. Com isso, o mesmo *gateway* estará presente em vários *clusters* virtuais, mas estará ativo em apenas um deles.

O problema desse esquema é que cada *cluster* terá um endereço IP virtual e cada hos-

pedeiro deverá ser configurado manualmente para se comunicar com um desses IPs, o que torna a manutenção bastante trabalhosa, talvez impossível em um ambiente com dispositivos IoT. Além disso, o recurso de preempção deverá ser configurado para o *gateway* ativo de cada *cluster* virtual, o que causará mais tempo de indisponibilidade para os dispositivos que estiverem conectados ao *cluster* do *gateway* ativo que falhou.

Já o protocolo GLBP possui um AVG (*Active Virtual Gateway*) e até 4 AVF (*Active Virtual Forwarder*). O primeiro será o *gateway* ativo, responsável por responder às requisições feitas ao endereço IP do grupo GLBP. Quando um novo hospedeiro deseja se comunicar com o grupo GLBP, o AVG aceita a solicitação enviando para o hospedeiro o endereço MAC virtual de um AVF. O grupo GLBP pode ter até quatro endereços MAC virtuais e o AVG pode ser também um AVF. Então a ideia é semelhante à configuração manual utilizada com o HSRP ou VRRP, mas utilizando-se de endereços MAC virtuais e não IP.

Por exemplo, em um grupo GLBP, o AVG é também um AVF e há outro *gateway* em espera (*standby*) configurado como segundo AVF. Parte dos hospedeiros da rede local se comunicam através do AVG, utilizando o IP virtual do grupo e o MAC virtual do primeiro AVF, enquanto a outra parte se comunica com o segundo AVF, utilizando o IP virtual do grupo e o MAC virtual do segundo AVF. Se o AVG falhar, o *gateway* em espera assume a função de AVG e de primeiro AVF, passando a responder requisições de toda a rede local.

2.11.4 *Common Address Redundancy Protocol* (CARP)

O protocolo *open source* CARP foi criado pela OpenBSD no início dos anos 2000 como uma opção ao VRRP, que estava no meio da polêmica de patentes da Cisco. Seu principal objetivo é prover redundância, mas possui um diferencial: foi concebido visando garantir segurança ao grupo de redundância que é criado. Assim como o VRRP, há um *gateway* mestre e um ou mais de *backup*. O mestre responde às requisições que chegam no endereço IP virtual do grupo de redundância e, caso sofra uma falha, um dos *gateways* de *backup* assume seu papel, de acordo com a prioridade definida em cada um (OPENBSD, 2019).

Os anúncios enviados pelo mestre são criptografados, dificultando a entrada de *gateways* maliciosos. Durante a configuração do CARP em cada *gateway*, uma senha é definida. Esta senha é utilizada para descriptografar os pacotes de anúncio.

Por conta do VRRP ter continuado livremente disponível, o protocolo CARP não se tornou tão popular e possui menos documentação disponível na Internet.

Capítulo 3

Proposta: Gateway IoT Redundante com uso de Fog Computing

3.1 Introdução

Uma rede IoT costuma ser composta de dispositivos que utilizam os mais diversos protocolos de comunicação, como HTTP, TCP, UDP, MQTT, *Bluetooth*, *Zigbee*, entre outros. Para que todos os dados sejam armazenados num *middleware*, o *gateway* IoT precisa se comunicar com os dispositivos SA através de todos esses protocolos a fim de receber os dados, conforme é mostrado na figura 3.1.

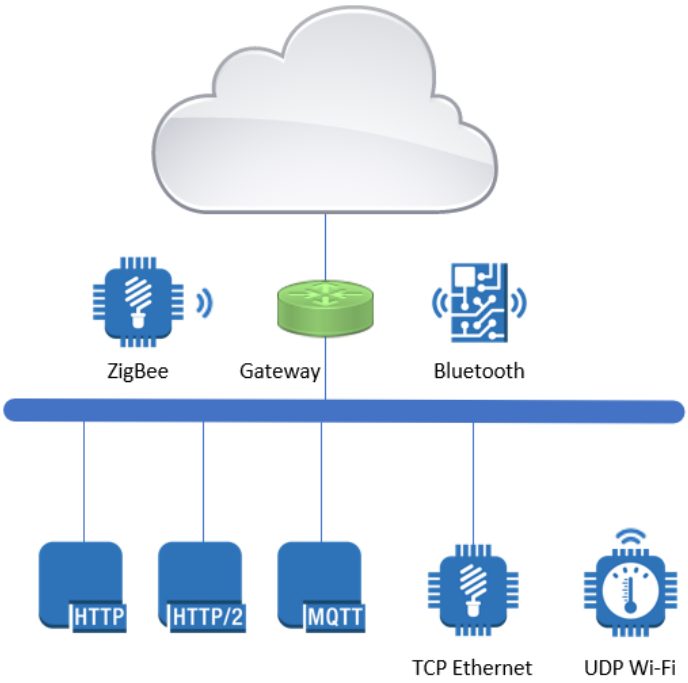

Figura 3.1: Rede IoT com vários protocolos de comunicação em uso.

Entretanto é fácil perceber pela imagem que uma função tão importante da rede está num único dispositivo. Se uma falha ocorrer nele, todos os dispositivos não terão mais como se comunicar com o *middleware* IoT.

A intenção deste trabalho é propor modificações no *gateway* IoT para prover alta disponibilidade na borda da rede, ao mesmo tempo em que parte da inteligência da nuvem seja estendida para mais perto dos dispositivos IoT, o que permite reações mais rápidas aos dados do ambiente, aproveita de uma maneira melhor os recursos ociosos disponibilizados após inserção da redundância, garante certo nível de independência da Internet e reduz o uso da banda disponível, enviando para a nuvem as informações de maneira condensada e menos prolixa.

3.2 *Hardware*

Um **Raspberry Pi** pode ser definido como um mini computador. É uma placa baseada em microprocessador, que possui todas as funções mais básicas de um computador, sendo possível rodar um sistema operacional, permitindo a execução de múltiplos programas simultaneamente, além de suportar o uso de diferentes linguagens de programação.

Por ter características de um computador, um Raspberry Pi possui portas USB, para conectar periféricos como mouses e teclados, e porta HDMI, para conectar um monitor. Também possui portas de entrada e saída para aplicações genéricas envolvendo sensores e atuadores. A placa já vem com suporte à comunicação através de cabo *Ethernet*, Wi-Fi e *Bluetooth*. Há vários modelos de Raspberry Pi e módulos extras para adicionar novas funções às placas.

Por ser relativamente barato, seu uso como protótipo se justifica em aplicações mais complexas que envolvem comunicação pela rede, execução de *softwares* dentro de um sistema operacional e o processamento e armazenamento de dados.

Para atuar como *gateway* IoT da rede local, foram utilizadas unidades do Raspberry Pi 3 Model B V1.2. Por conter um processador Cortex-A53 BCM2837, esse modelo suporta arquiteturas 64-bit (BROWN, 2016). Para montar o esquema de redundância, dois Raspberry Pi desse modelo foram configurados praticamente da mesma maneira. A figura 3.2 mostra as duas unidades.

Há um pendrive de 8 GB conectado em cada um para armazenar os bancos de dados com informações dos dispositivos IoT da rede. Durante testes, foi observado que as altas taxas de escrita nos cartões de memória micro-SD acabavam reduzindo rapidamente sua vida útil ou causando constantes problemas de corrupção de dados do sistema. Por isso, optou-se pelo uso de um dispositivo externo para servir como espaço de armazenamento para o banco de dados numa mídia separada da que possui o sistema operacional. Caso haja necessidade, é possível substituir o pendrive por um HD ou SSD em conjunto com um adaptador SATA-USB com fonte, visando mais espaço de armazenamento ou taxas maiores de escrita.

As duas placas Raspberry Pi foram configuradas para se conectar à rede apenas através do Wi-Fi. Uma conexão cabeada seria mais robusta e menos sujeita a falhas envolvendo o AP da rede sem fio, mas, para a apresentação, optou-se por remover os cabos para simplificar a montagem. O uso de uma conexão *Ethernet*, entretanto, é possível e bastaria realizar adaptações específicas na configuração do programa *Keepalived*, de modo que o endereço IP virtual do VRRP fosse utilizado na interface *Ethernet* eth0 do Raspberry Pi.

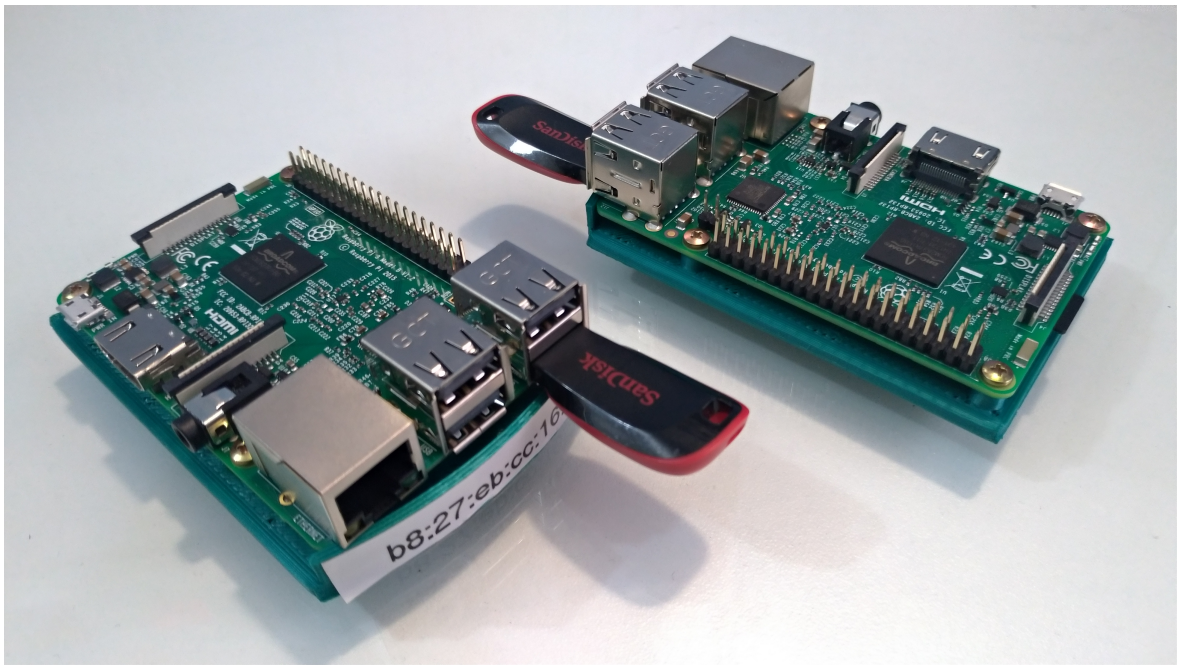

Figura 3.2: Dois Raspberry Pi 3 Model B V1.2 utilizados para prototipação e demonstração

3.3 *Softwares*

Alguns *softwares open source*, ou seja, de código aberto, foram utilizados em conjunto com *softwares* desenvolvidos pelo laboratório de Internet das Coisas *UnB Internet of Things* - UIoT, da Universidade de Brasília - UnB. Esses *softwares* do UIoT ainda estão em desenvolvimento e podem não estar abertos ao público. Tais programas serão apresentados a seguir.

3.3.1 Sistema Operacional *Debian*

Dentre os diversos sistemas operacionais disponíveis para o Raspberry Pi 3, optou-se pelo uso de um que utilizasse arquitetura 64-bit, para tirar maior proveito do banco de dados *MongoDB*. Em geral, os sistemas operacionais estáveis que estão disponíveis para o Raspberry Pi atualmente são baseados em 32-bit, apesar de haver vários projetos em desenvolvimento de imagens baseadas em 64-bit.

Então foi utilizada uma imagem de um *Debian Buster* para Raspberry Pi 3, uma versão em desenvolvimento do *Debian*, com suporte à arquitetura 64-bit. Essa versão não está totalmente estável (DEBIAN, 2019), então ainda não é recomendada para ambientes de produção, mas foi utilizada para a prototipagem da proposta.

3.3.2 UIoT *Middleware*

O ***UIoT Middleware*** é um conjunto de *softwares* que englobam tanto a função de *gateway* IoT quanto de processamento de dados (UIOT, 2019). As características do *UIoT Middleware* são:

1. Ter suporte aos protocolos HTTP, TCP, UDP, MQTT e *Zigbee*, além de estar em desenvolvimento para atender também *Bluetooth* e GPRS.

2. Atender as requisições feitas pelos dispositivos IoT a fim de recolher dados sobre o ambiente.

3. Armazenar e processar dados recebidos pelos dispositivos IoT de uma rede.

4. Exibir e classificar dados IoT recebidos e armazenados para facilitar a interpretação por pessoas.

5. Repassar dados para outros *middlewares* e fazer requisições a dispositivos SA.

6. Realizar o registro de clientes, serviços e dados para criar uma camada de segurança no serviço.

O *UIoT Middleware* será responsável por aplicar a arquitetura de *Fog Computing* na rede IoT. Com essas funções, será possível:

1. Realizar análise de dados e tratar dados sensíveis à latência de maneira mais eficaz, acionando atuadores da rede a partir do *gateway* IoT.

2. Condensar dados enviados pelos dispositivos SA para economizar banda ao repassar os dados à nuvem.

3. Dar autonomia à rede local, mantendo os serviços e o armazenamento de dados mesmo que a conexão com a Internet caia.

De acordo com (FERREIRA et al., s.d.), o *UIoT Middleware* tem considerações sobre escalabilidade, transparência, diversidade, mobilidade, performance, facilidade de uso, expansibilidade e é desenvolvido de maneira *open source* e com tecnologias conhecidas.

3.3.3 *Keepalived*

Dentre as opções disponíveis de protocolos de redundância mostradas na seção 2.4, HSRP e GLBP eram protocolos proprietários e não poderiam ser utilizados no protótipo. Entre o CARP e o VRRP, o segundo possui mais documentação e informações disponíveis na Internet, o que facilitaria o desenvolvimento do projeto. Por isso o VRRP foi o protocolo de redundância escolhido.

O *Keepalived* é um *software* que serve para oferecer alta disponibilidade através do VRRP, além de balanceamento de carga. Ele pode ser instalado em dispositivos trabalhando com um sistema operacional Linux. A carga, neste caso, é composta de dados que são enviados pela rede para o dispositivo que está executando o *Keepalived* (KEEPALIVED, 2018). Apesar de se proclamar simples, ele é um programa com várias opções de configuração que não são tão triviais, o que pode dificultar até mesmo a montagem de ambientes pequenos.

O balanceamento de carga é obtido pelo uso de um *Linux Virtual Server*, ou LVS, ou seja, dois ou mais servidores Linux físicos formam um *cluster*, o qual é considerado um único servidor virtual, o LVS (WENSONG, 2011). Como a carga está localizada na camada

de rede, o *Keepalived* faz uso do *software IP Virtual Server*, ou IPVS, para realizar sua distribuição. Quando dados e requisições chegam no LVS, eles são redirecionados para os servidores físicos pelo IPVS, de maneira que estes se revezam, dificultando assim que um deles se sobrecarregue. Outro benefício é a simplificação da topologia lógica da rede (WENSONG, 2016).

A alta disponibilidade é oferecida com o uso do VRRP, que funciona da mesma maneira como foi explicado na subseção 2.11.2: um dos servidores assume o IP virtual do *cluster* e passa a responder por todas as requisições feitas a esse IP. Se o servidor mestre ficar indisponível, o servidor *backup* se torna o novo mestre e passa a responder as requisições feitas ao IP virtual do *cluster*.

Para identificar a falha nos dispositivos, o *Keepalived* utiliza o protocolo *Bidirectional Forwarding Detection* (protocolo BFD), que é eficiente e rápido, o que garante menos tempo de indisponibilidade do que outras soluções (IETF, 2010b).

Para o trabalho proposto, apenas o recurso de alta disponibilidade, ou seja, o uso do VRRP, será utilizado. O balanceamento de carga utilizando o próprio *Keepalived* ficará como parte dos trabalhos futuros.

Por ter sido lançado sob a GNU *General Public License*, o *Keepalived* é um *software open source*, que pode ser utilizado livremente.

3.3.4 *MongoDB*

Durante o desenvolvimento do *software* do *UIoT Middleware*, foi utilizado um arquivo .txt para salvar as informações dos dados recebidos pelos dispositivos IoT da rede local. Tal recurso foi utilizado simplesmente para validação de fluxo de funcionamento do *software* e testes de integração com os *middlewares* IoT.

Foi, então, necessário escolher um banco de dados para oferecer segurança às informações obtidas do ambiente e de dispositivos, além de conceder maior facilidade para manipular e armazenar esses dados. Dentre as opções *open source* disponíveis, o *MongoDB* foi escolhido pelo time de desenvolvimento do laboratório UIoT e os dados são agora armazenados no formato JSON.

MongoDB é um banco de dados orientado a documentos, *open source* e gratuito (MONGODB, 2019). Por ser flexível e escalável, é uma ótima alternativa para soluções que envolvem grandes volumes de dados que podem ser bastante distintos entre si (por exemplo, medições de temperatura ambiente e o estado de uma lâmpada).

Uma desvantagem é que o *MongoDB* em 32-bit é limitado a 2 GB de dados (MONGODB, 2009), o que pode ser facilmente alcançado em ambientes IoT relativamente complexos. Com o sistema operacional 64-bit, essa limitação é maior e deixa de ser uma preocupação no momento.

3.4 Arquitetura

Utilizando impressão 3D, foi possível criar um compartimento capaz de acomodar o Raspberry Pi 3, conectado a um botão que liga e desliga a placa e um LED colorido indicando

seu estado atual (mestre para a cor verde e *backup* para a cor roxa). A imagem do protótipo final pode ser vista na figura 3.3.

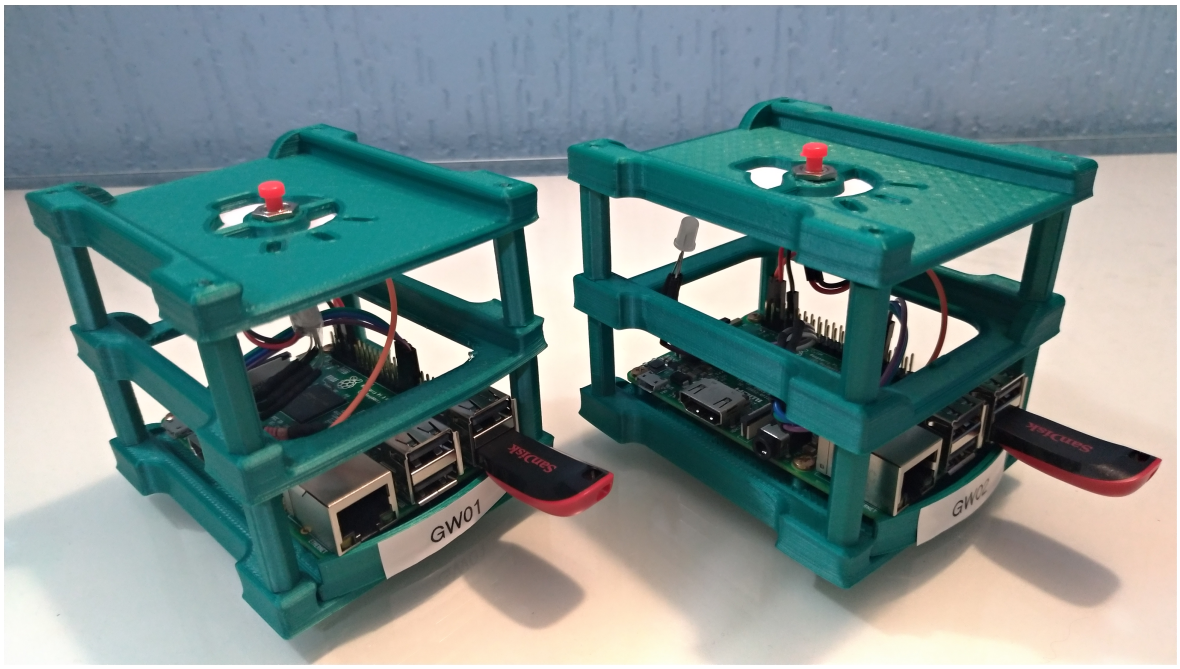

Figura 3.3: Apresentação do protótipo final.

A arquitetura desenvolvida nesse trabalho (figura 3.4) define os dispositivos SA como sendo responsáveis pela coleta de dados de equipamentos e do ambiente, fazendo parte da *Mist Computing*. Os *gateways* IoT redundantes estão na borda da rede com um *middleware* IoT em execução, aplicando *Fog Computing* ao receber, armazenar, repassar e processar os dados dos dispositivos SA. Com a redundância, há alta disponibilidade de distribuição das funções do *middleware* entre os *gateways*. Por fim, há um *middleware* em execução na nuvem também (*Cloud Computing*), onde uma quantidade maior de dados pode ser processada, mas havendo uma latência maior na comunicação com os dispositivos SA.

Nos Anexos I, há um passo a passo de como configurar um Raspberry Pi 3 desde o início para que ele atue como um *gateway* IoT, mas alguns programas podem não estar acessíveis publicamente, como, por exemplo, o *UIoT Middleware* que está em um repositório fechado no momento da realização desse trabalho.

3.5 Funcionamento

As duas placas Raspberry Pi 3 possuem, entre outros, os seguintes *softwares* instalados, configurados e em execução desde a inicialização do sistema operacional: *UIoT Middleware*, *MongoDB* e *Keepalived*. Estes são os principais programas para que a proposta ofereça redundância e *Fog Computing*.

O *Keepalived* é o responsável por disponibilizar o endereço IP virtual (VIP) que estará presente em alguma das duas placas por vez. O VIP aparecerá apenas na interface *wireless*, chamada no *Debian* de wlan0. A placa que tiver o VIP em sua interface wlan0 será denominada de ***Gateway* Mestre**, enquanto que a que não tiver o VIP será denominada de ***Gateway Backup***.

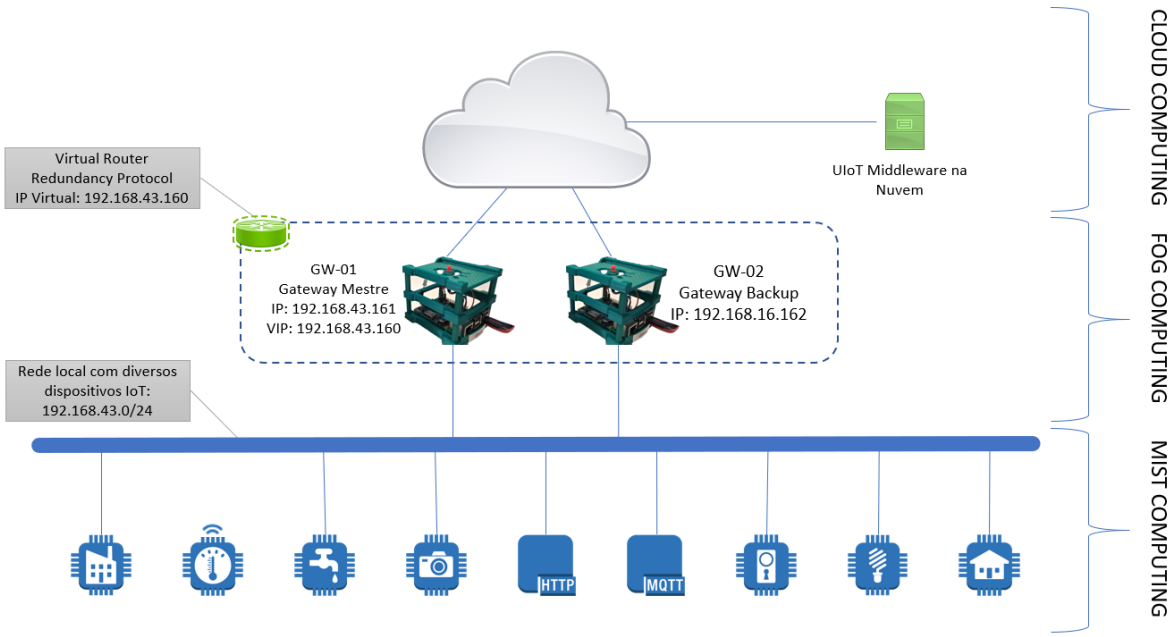

Figura 3.4: Arquitetura proposta.

Cada placa possui um *hostname* e um endereço IP fixo: o GW-01 possui o endereço IP 192.168.43.161 e o GW-02 possui o endereço IP 192.168.43.162. O VIP escolhido foi o 192.168.43.160 e esse será o *gateway* padrão configurado nos dispositivos IoT da rede local para enviar os dados.

As placas funcionam da mesma maneira como foi explicado anteriormente. Se o *Gateway* Mestre atual sofrer uma falha e perder conexão com a rede, o *Gateway Backup* assumirá o VIP em sua interface wlan0 e passará a ser o *Gateway* Mestre.

A opção de preempção está ativada para manter o GW-01 como mestre e o GW-02 como *backup* sempre que os dois estiverem disponíveis. Ou seja, se o GW-01 sofrer uma falha e perder conexão com a rede, o GW-02 assumirá o papel de *Gateway* Mestre utilizando o VIP em sua interface de rede sem fio. Quando o GW-01 se recuperar da falha e voltar a ter conexão com a rede, ele assumirá novamente o VIP, fazendo com que o GW-02 volte a ser um *Gateway Backup*.

Com o *software UIoT Middleware* rodando, a porta 8000 será utilizada para receber dados via HTTP dos dispositivos finais. Como as requisições serão enviadas para o VIP, sempre será o *Gateway* Mestre do momento que receberá os dados. Para manter os bancos de dados dos dois *gateways* sendo populados, os dados recebidos pelo mestre serão enviados para o seu próprio *UIoT Middleware* local, para o *UIoT Middleware* do *backup* e para o *UIoT Middleware* hospedado na nuvem.

Desse modo, o *Gateway* Mestre atuará mais como *gateway* de comunicação. O *Gateway Backup* terá como funções principais o armazenamento redundante, o processamento dos dados novos e servir para exibir os dados do banco de dados pelo *UIoT Middleware*, desafogando assim o *Gateway* Mestre dessas funções.

Nas subseções a seguir, os cenários possíveis serão descritos, explicando o que aconteceria com cada *gateway*, qual seria seu estado e qual seria o fluxo dos dados.

3.5.1 Cenário sem falhas

Nesse cenário, há um *Gateway* Mestre e um *Gateway Backup*. Para exemplificar, inicialmente o GW-01 é o *Gateway* Mestre e o GW-02 é o *Gateway Backup*. Isso significa que o GW-01 possui os endereços IP 192.168.43.161 e VIP 192.168.43.160 em sua interface *wireless* wlan0, enquanto o GW-02 possui apenas o IP 192.168.43.162.

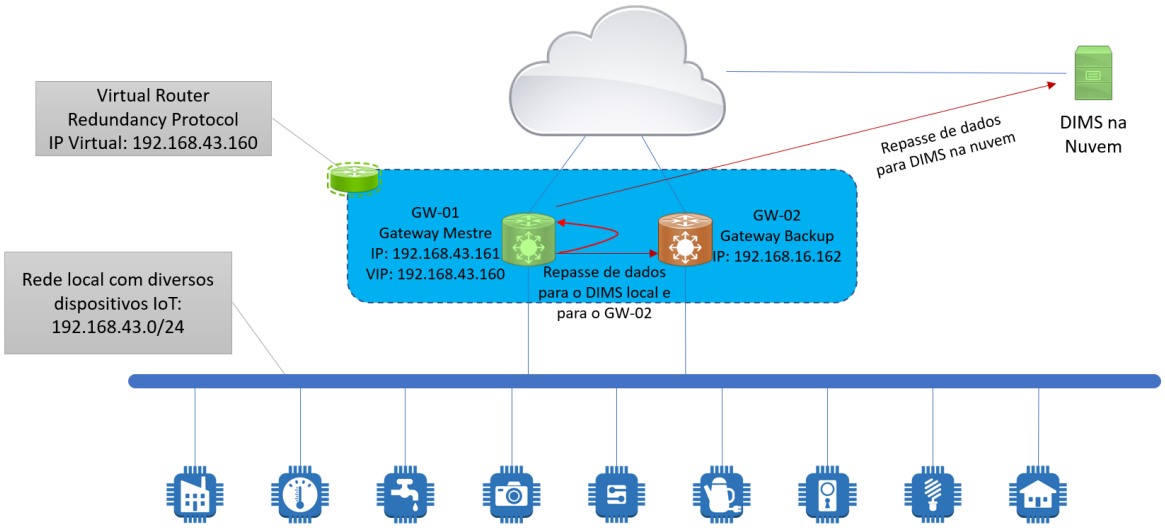

Figura 3.5: Cenário sem falhas.

Os dados de algum dispositivo IoT são recebidos pelo *Gateway* Mestre (GW-01) através do VIP e da porta 8000 (HTTP). Esses dados são repassados para o seu próprio *UIoT Middleware* e, então, armazenados no banco de dados *MongoDB* do GW-01. Eles também são repassados para o *UIoT Middleware* do GW-02 e para o *UIoT Middleware* hospedado na nuvem. O GW-02, por sua vez, realizará o processamento dos novos dados.

3.5.2 Cenário com falha no *Gateway Backup*

Partindo do cenário sem falhas, caso o *Gateway Backup* (GW-02) perca conexão com a rede, os dados não poderão mais ser processados por ele. Nesse caso, o *Gateway* Mestre (GW-01) continuará funcionando independente do estado do GW-02. O GW-01 enviará os dados seu *UIoT Middleware* local e para o da nuvem. Ele mesmo também processará os dados recebidos.

Caso o GW-02 se recupere da falha e volte a se conectar à rede, o GW-01 volta a repassar os dados para ele, que poderá continuar processando os novos dados.

3.5.3 Cenário com falha no *Gateway* Mestre

Partindo do cenário sem falhas, caso o *Gateway* Mestre (GW-01) perca conexão com a rede, o GW-02 muda de estado de *Gateway Backup* para *Gateway* Mestre, assumindo o VIP 192.168.43.160 junto com seu IP 192.168.43.162 em sua interface de rede sem fio. Como não há outro *Gateway Backup* disponível, o GW-02 repassará os dados para o seu próprio

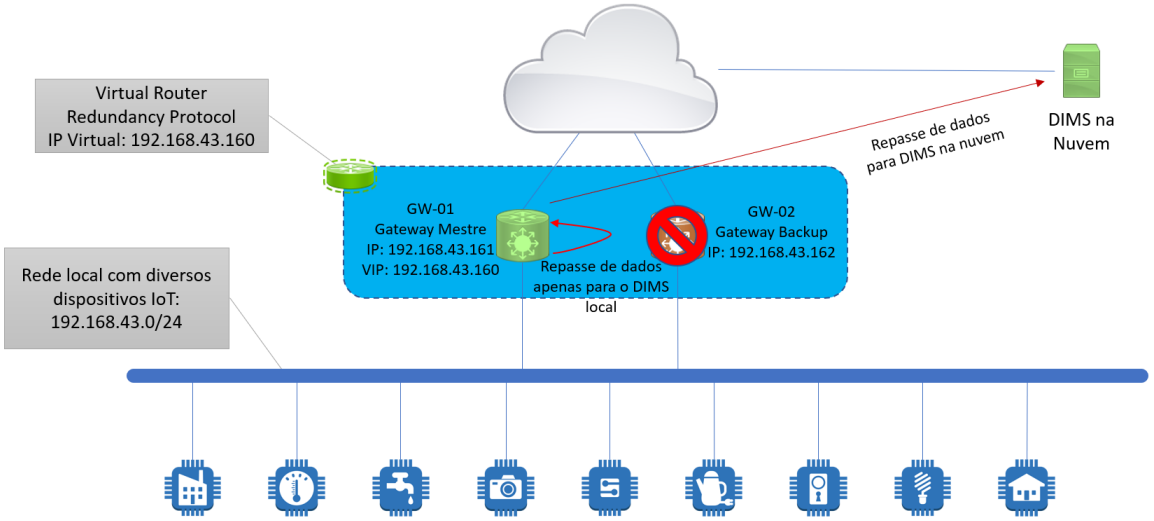

Figura 3.6: Cenário com falha no *Gateway Backup*.

UIoT Middleware e para o da nuvem. Ele também continuará processando os novos dados recebidos.

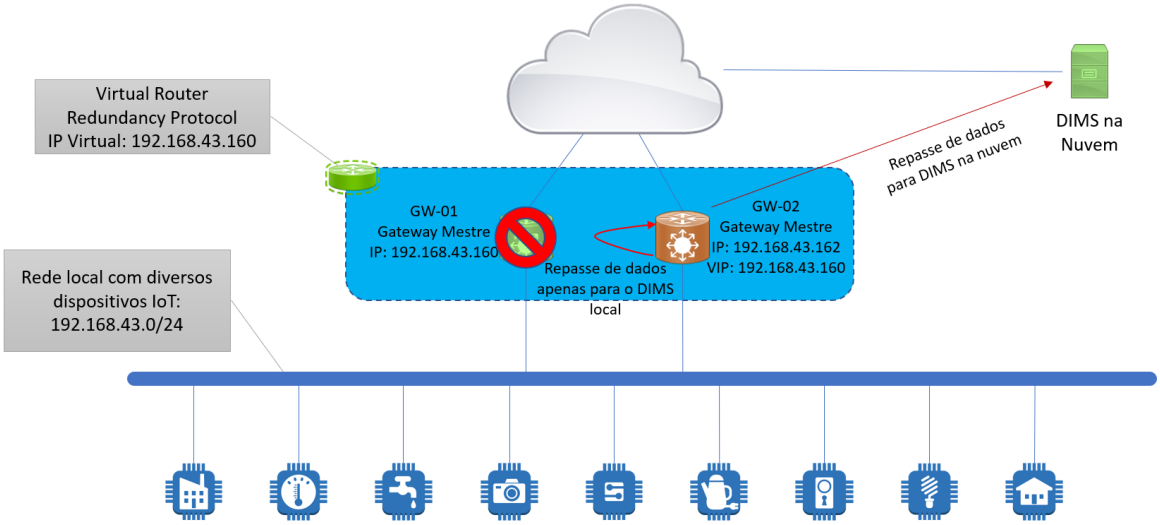

Figura 3.7: Cenário com falha no *Gateway* Mestre

Caso o GW-01 se recupere da falha e volte a se conectar à rede, por conta de a opção de preempção estar habilitada, o GW-01 assume novamente o papel de *Gateway* Mestre, voltando a atuar como *gateway* de comunicação, enquanto o GW-02 volta a focar no processamento e exibição de dados. O GW-02, então, volta ao estado de *Gateway Backup*, retornando assim para o cenário inicial sem falhas.

Capítulo 4

Resultados

Nesse capítulo, serão demonstrados os resultados obtidos após a realização de testes dos cenários apresentados nas subseções 3.5.1, 3.5.2 e 3.5.3. Para isso, as simulações foram feitas utilizando os protótipos desenvolvidos e um *script* que envia dados como se fosse um sensor de temperatura do ambiente.

No caso do cenário sem falhas, o funcionamento e o desempenho do *middleware* com dois *gateways* foram analisados, com foco na distribuição da carga de processamento.

Durante o cenário com falha no *Gateway Backup*, foi observado como o *Gateway* Mestre passa a realizar o processamento de dados ao perceber que o *backup* não estava disponível.

Por fim, foi visto como o *Gateway Backup* muda de estado em caso de falha no *Gateway* Mestre, mantendo ativas todas funções do *gateway* IoT da rede.

4.1 Cenário sem falhas

Na figura 4.1 abaixo, é possível ver que o GW-01 possui o endereço IP virtual (VIP) 192.168.43.160, logo ele é o *Gateway* Mestre, enquanto que o GW-02 é o *Gateway Backup*.

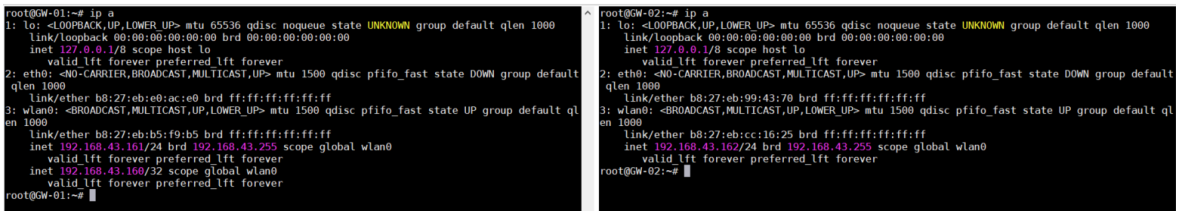

Figura 4.1: Cenário sem falhas - Endereços IP dos *gateways*.

Na figura 4.2, é mostrado o banco de dados de clientes registrados nos *gateways*. Os bancos de dados estão inicialmente vazios, ou seja, nenhum cliente foi registrado ainda. Na parte superior, a consulta é feita diretamente ao banco de dados, enquanto que na parte inferior, os dados podem ser visualizados através do *UIoT Middleware*.

Na figura 4.3, utilizando uma máquina virtual rodando Ubuntu, uma requisição HTTP na porta 8000 do VIP foi feita, utilizando uma ferramenta chamada *curl*, simulando um novo dispositivo IoT desejando se comunicar com o *gateway* IoT para posteriormente enviar informações sobre seus serviços.

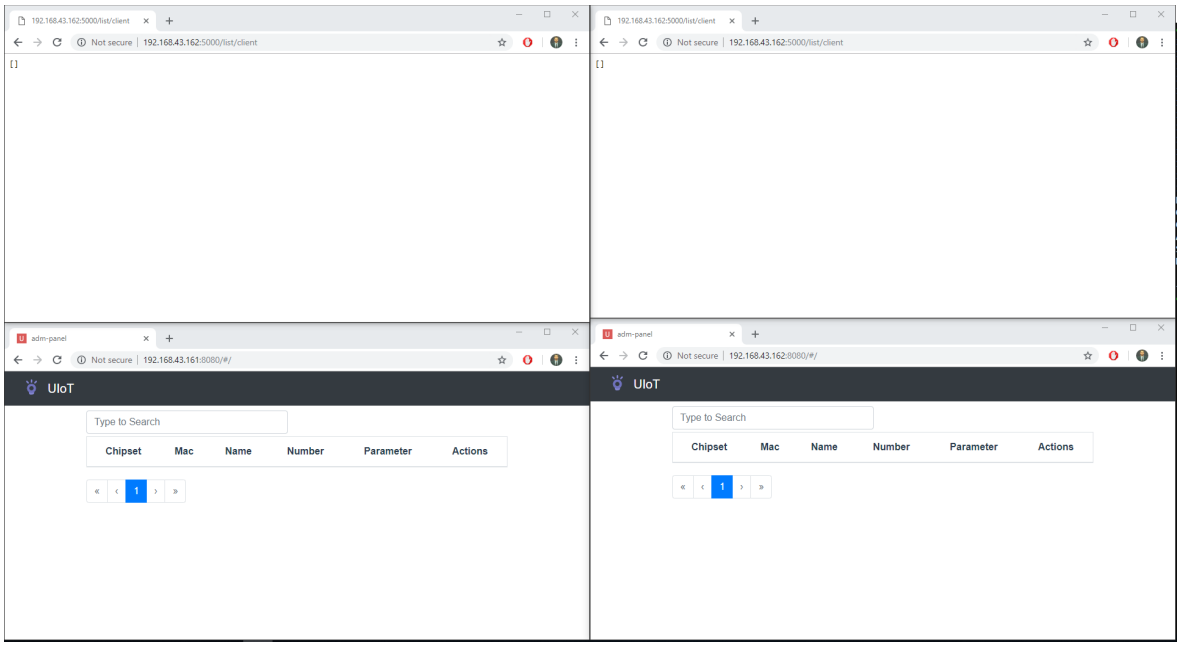

Figura 4.2: Cenário sem falhas - Estado inicial do banco de dados dos *gateways*.

```
cassiofabius@CassioFabiusDell7559:~$ curl -i --request POST \
>      --url http://192.168.43.160:8000/client \
>      --header 'content-type: application/json' \
>      --data '{
>          "clientTime": 1000000000.1111,
>          "tags": [ "TESTE-tag" ],
>          "name": "Arduino-Uno-Teste",
>          "chipset": "Atmega8U2",
>          "mac": "FF:FF:FF:FF:FF:FF",
>          "serial": "C210",
>          "processor": "ATmega328",
>          "channel": "Ethernet",
>          "location": "-15.7757876:-48.077829"
>      }'
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 33
Access-Control-Allow-Origin: *
Server: Werkzeug/0.14.1 Python/3.7.2+
Date: Fri, 15 Feb 2019 09:23:30 GMT

{"code":200,"message":"Success"}
cassiofabius@CassioFabiusDell7559:~$
```

Figura 4.3: Requisição HTTP para cadastro de um novo cliente.

Atualizando as páginas que mostravam o conteúdo dos bancos de dados, um novo cliente surgiu no banco de dados dos *gateways*, como mostra a imagem 4.4. Como esperado, a requisição foi feita para o VIP, que estava no GW-01, o *Gateway* Mestre. Ele recebeu os dados do cliente e repassou para o seu próprio *UIoT Middleware*, para o *UIoT Middleware* do GW-02 e para o *UIoT Middleware* na nuvem.

Não há dados a serem exibidos pelo *UIoT Middleware* ainda porque apenas serviços e informações são mostrados. Como apenas o cliente foi cadastrado, as janelas do *middleware* continuam vazias.

Agora é necessário cadastrar um novo serviço desse cliente. O cliente cadastrado foi simulado como se fosse um Arduino Uno conectado à rede via Ethernet, que pode ter vários

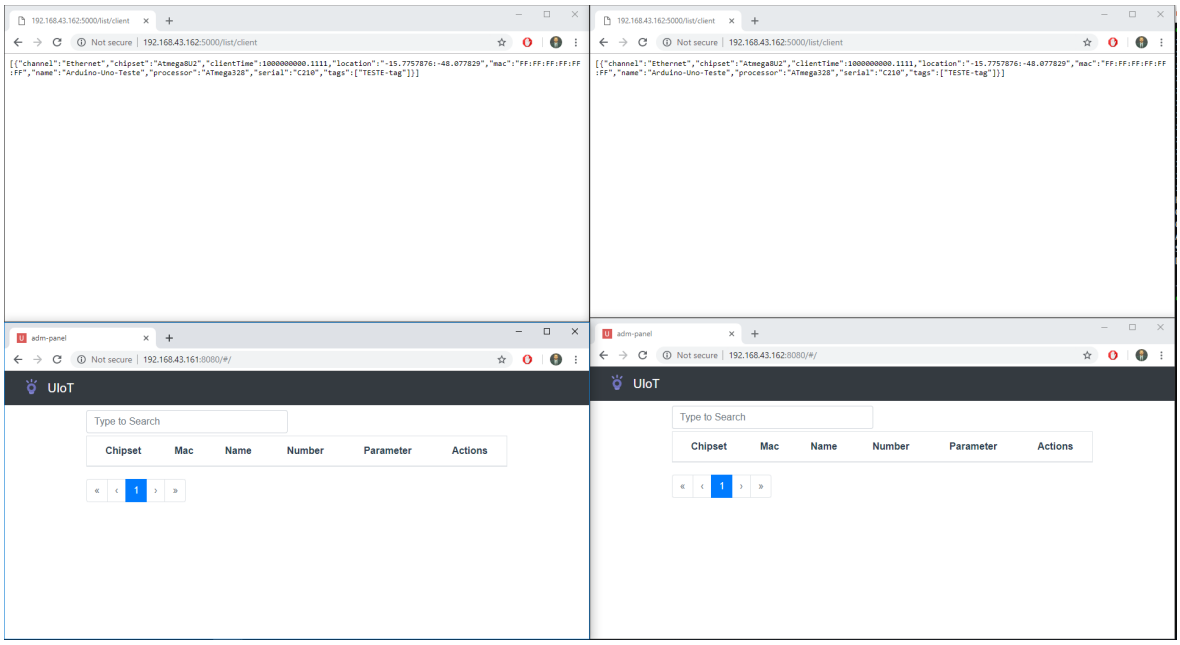

Figura 4.4: Bancos de dados dos dois *gateways* com um cliente cadastrado.

sensores, como um DHT11 que lê a temperatura e a umidade do ambiente. Inicialmente, o banco de dados de serviços está vazio, como pode ser visto na figura 4.5.

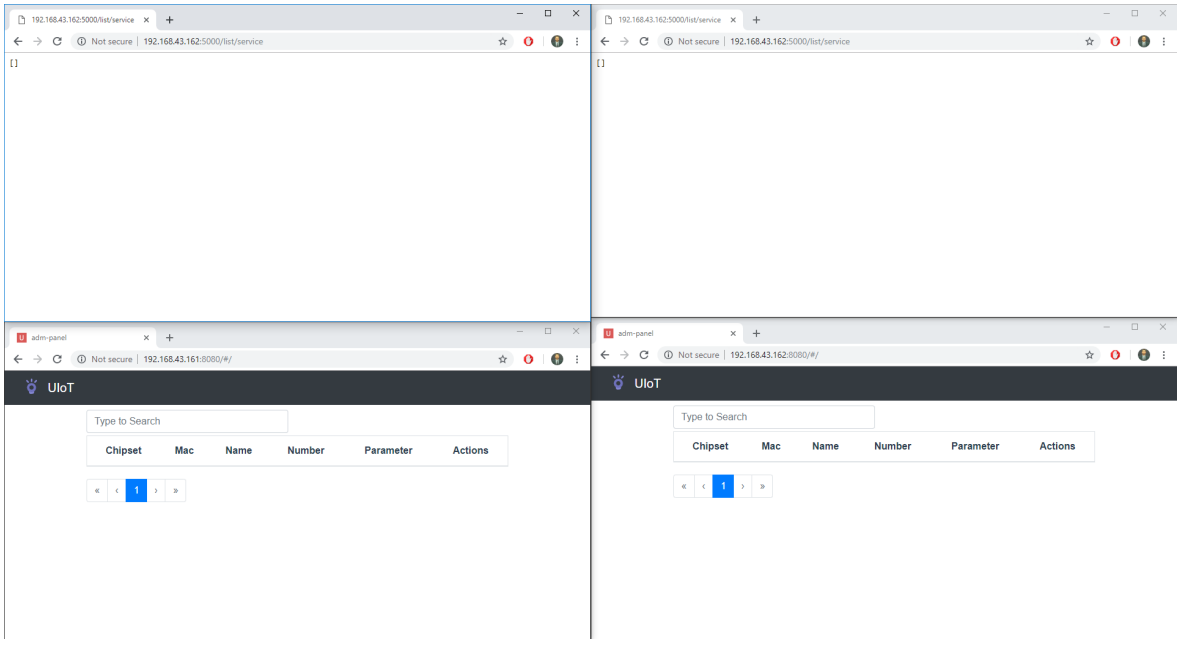

Figura 4.5: Banco de dados de serviços vazio nos dois *gateways*.

Uma nova requisição HTTP é feita ao *gateway*, mas dessa vez é para que um novo serviço seja cadastrado, a temperatura do ambiente que está sendo medida pelo sensor conectado ao Arduino Uno. A figura 4.6 mostra a requisição e a resposta recebida.

Os dados do novo serviço são inseridos no *UIoT Middleware* e os bancos de dados são sincronizados. Os dados podem ser vistos na parte superior da figura 4.7, mas também é possível ver essas informações através do *UIoT Middleware*, como está sendo mostrado na parte inferior.

```
cassiofabius@CassioFabiusDell7559:~$ curl -i --request POST \
>     --url http://192.168.43.160:8000/service \
>     --header 'content-type: application/json' \
>     --data '{
>         "clientTime": 1000000000.1,
>         "tags": [ "TESTE-tag" ],
>         "number": 3,
>         "chipset": "Atmega8U2",
>         "mac": "FF:FF:FF:FF:FF:FF",
>         "name": "Get temp",
>         "parameter": "temperature",
>         "unit": "ºC",
>         "numeric": 1
>     }'
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 33
Access-Control-Allow-Origin: *
Server: Werkzeug/0.14.1 Python/3.7.2+
Date: Fri, 15 Feb 2019 09:26:56 GMT

{"code":200,"message":"Success"}
cassiofabius@CassioFabiusDell7559:~$
```

Figura 4.6: Requisição HTTP para cadastro de novo serviço.

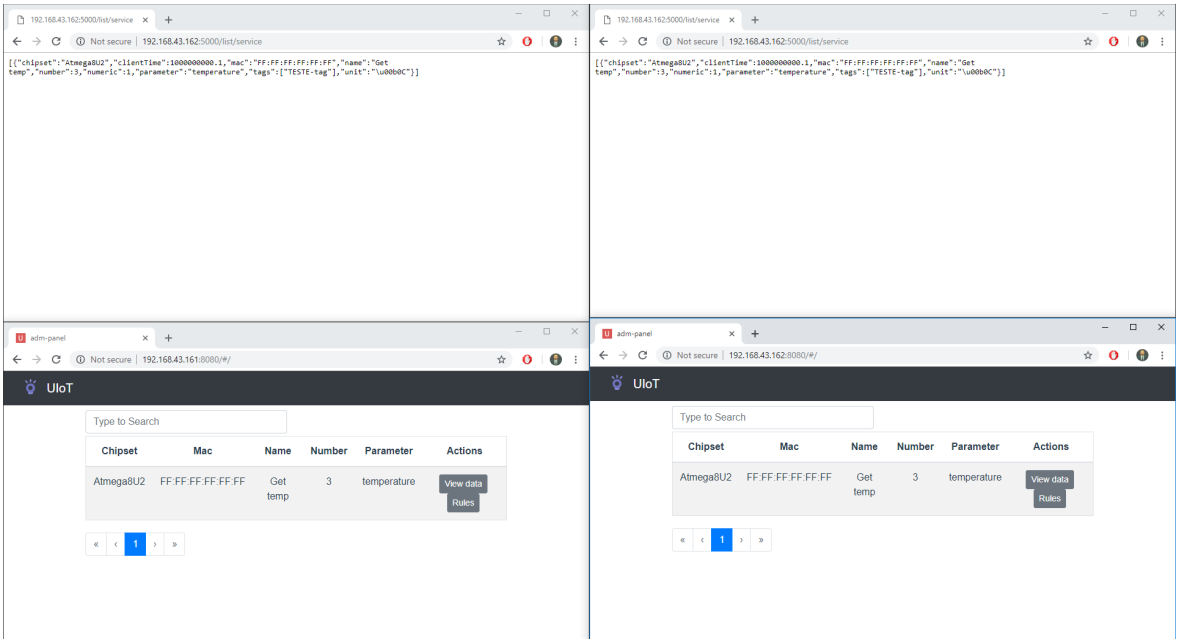

Figura 4.7: Dados do novo serviço presentes nos bancos de dados dos *gateways*.

Após o cadastro do cliente e de seu serviço, é possível começar a enviar dados relacionados às informações obtidas através de sensores no dispositivo IoT. A figura 4.8 mostra o banco de dados das informações inicialmente vazio.

Uma nova requisição utilizando *curl* é feita para enviar informações de temperatura para o *gateway* (figura 4.9).

Os dados no banco de dados nos dois *gateways* podem ser vistos na parte superior da figura 4.10, mas agora também é possível clicar "*view data*"da figura 4.7 e visualizar as informações cadastradas de acordo com a data e horário de cada uma e sua variação em um gráfico ao lado, como mostrado na figura 4.10.

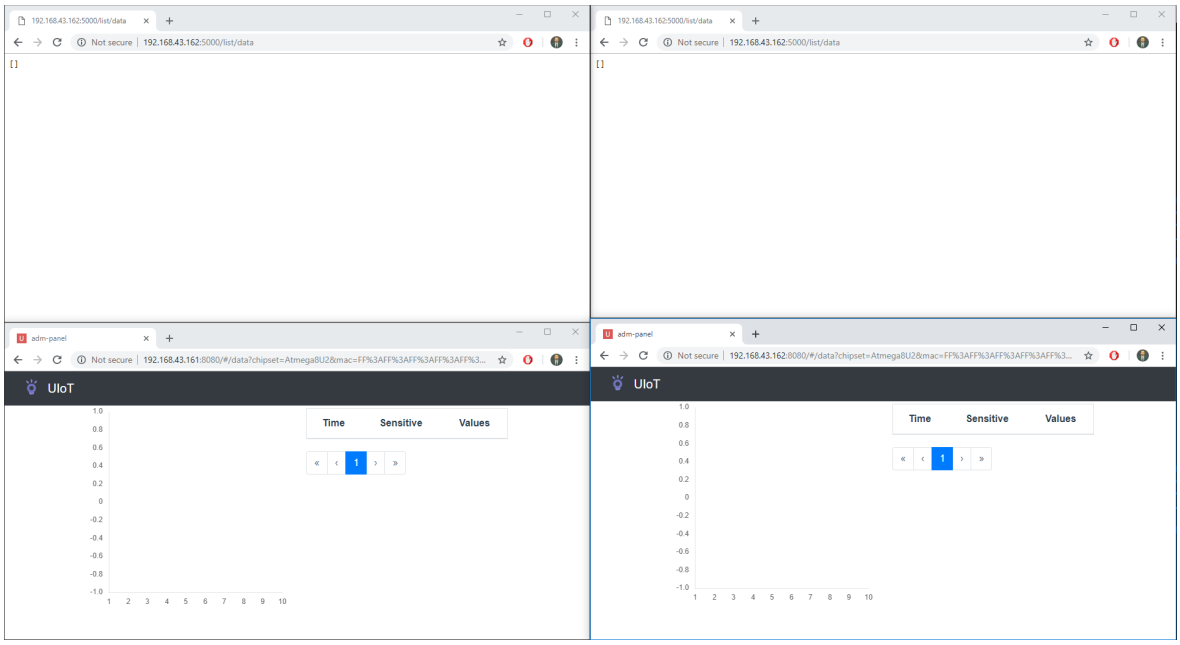

Figura 4.8: Banco de dados de informações vazio nos dois *gateways*.

```
cassiofabius@CassioFabiusDell7559:~$ curl -i --request POST \
>      --url http://192.168.43.160:8000/data \
>      --header 'content-type: application/json' \
>      --data '{
>          "clientTime": 1000000000.1,
>          "tags": [ "TESTE-tag" ],
>          "sensitive": 1,
>          "chipset": "Atmega8U2",
>          "mac": "FF:FF:FF:FF:FF:FF",
>          "serviceNumber": 3,
>          "values": [ "14.0" ]
>      }'
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 33
Access-Control-Allow-Origin: *
Server: Werkzeug/0.14.1 Python/3.7.2+
Date: Fri, 15 Feb 2019 09:29:12 GMT

{"code":200,"message":"Success"}
cassiofabius@CassioFabiusDell7559:~$
```

Figura 4.9: Nova requisição HTTP para envio de informações sobre o ambiente.

Além disso, a nova informação recebida pode ser processada e exibida de forma legível para o usuário. Cada informação terá um valor em geral numérico, que pode ser analisado e transformado em uma palavra ou ação por parte do *UIoT Middleware*. No caso da temperatura com valor de 14 °C, foi definido que esse valor é suficiente para definir que o ambiente encontra-se frio ("*cold*").

Esse processamento é feito somente no *Gateway Backup* (GW-02), na situação em que os dois *gateways* estiverem disponíveis. De maneira similar, se um valor de temperatura maior for enviado, o dado é considerado suficiente para concluir que o ambiente está quente

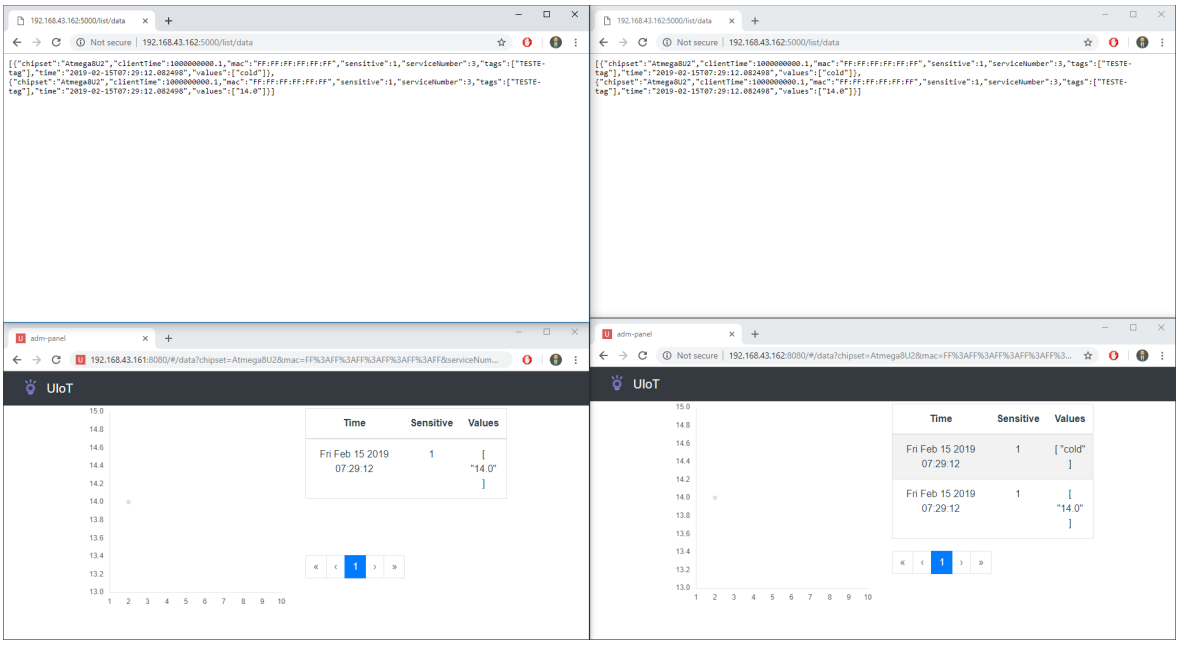

Figura 4.10: Banco de dados de informações populado com a nova informação recebida e a informação processada.

("*hot*").

4.1.1 Teste de Estresse do *UIoT Middleware*

A fim de testar o desempenho do *UIoT Middleware*, um teste de estresse foi realizado durante o cenário sem falhas nos *gateways*. Desse modo, será possível definir um fluxo de dados ideal para que o *middleware* em um Raspberry Pi 3 não seja sobrecarregado.

Para o teste de estresse em um cenário sem falhas, um *script* enviou 100 dados de um mesmo serviço, um a cada 500 ms, via protocolo HTTP ao *Gateway* Mestre. Foram feitas 5 rodadas de testes, sem que o banco de dados fosse resetado. Os dados tinham valores entre -20 e 40.

O objetivo é avaliar o desempenho do *UIoT Middleware* sendo executado em Raspberry Pi na borda da rede. Os bancos de dados dos *UIoT Middlewares* dos *gateways* locais e da nuvem foram comparados em questão de número de dados armazenados e a consistência deles.

A consistência dos dados poderia ser perfeita, caso todos os dados tivessem sido armazenados corretamente; boa, caso não houvesse nenhum dado repetido, mas com ocorrência de perda de dados; e ruim, caso dados tivessem sido perdidos e armazenados de maneira incorreta ou repetida.

Os resultados encontram-se na tabela 4.1 abaixo.

O *UIoT Middleware* na nuvem recebeu todos os dados com 100% de precisão nas quatro primeiras rodadas. Entretanto, na última rodada, algum problema ocorreu fazendo com que os 17 últimos dados não fossem entregues, apesar de todos terem chegado ao GW-01 com sucesso, o que fez com que a consistência dos dados fosse considerada boa.

UIoT Middleware	Rodada	1	2	3	4	5
GW-01	nº de dados	87	86	91	87	75
	Consistência	Ruim	Ruim	Ruim	Ruim	Ruim
GW-02	nº de dados	176	180	184	178	154
	Consistência	Ruim	Boa	Ruim	Ruim	Ruim
Nuvem	nº de dados	100	100	100	100	83
	Consistência	Perfeita	Perfeita	Perfeita	Perfeita	Boa

Tabela 4.1: Resultados do teste de estresse do *UIoT Middleware* nos *gateways* para envio de dados com intervalo de 500ms.

Nas quatro primeiras rodadas, o *UIoT Middleware* no GW-01 conseguiu repassar os dados para a nuvem perfeitamente, mas teve problemas na última rodada. Em todas as rodadas, dados foram deixados de ser armazenados e parte dos dados foi repetida, o que fez com que a consistência dos dados fosse ruim.

O *UIoT Middleware* no GW-02 conseguiu processar com sucesso todos os dados que armazenou, ou seja, definiu se o ambiente estava quente ou frio baseado no valor recebido, mas, assim como o GW-01, teve problemas ao armazená-los, por vezes repetindo dados ou simplesmente perdendo informações. Na segunda rodada, apesar de não ter armazenado alguns dados, não havia repetições, ou seja, todos dados tinham sido corretamente armazenados.

Realizando um novo teste de estresse, mas dessa vez com o envio de um dado por segundo, o *UIoT Middleware* se comportou muito melhor. Sempre 100 dados eram armazenados pelos três *middlewares*, apesar de muitos dados repetidos terem sido observados na primeira rodada de testes. Os dados eram repetidos pois a *timestamp* do dado em JSON recebido era a mesma para dois ou mais itens do banco de dados. Aparentemente o *UIoT Middleware* consegue perceber que um novo dado foi recebido, mas não armazena o dado novo e sim um antigo ainda presente em algum *buffer*. Os resultados podem ser vistos na tabela 4.2 abaixo.

UIoT Middleware	Rodada	1	2	3	4	5
GW-01	nº de dados	100	100	100	100	100
	Consistência	Ruim	Perfeita	Perfeita	Perfeita	Perfeita
GW-02	nº de dados	200	200	200	200	200
	Consistência	Ruim	Perfeita	Perfeita	Perfeita	Perfeita
Nuvem	nº de dados	100	100	100	100	100
	Consistência	Perfeita	Perfeita	Perfeita	Perfeita	Perfeita

Tabela 4.2: Resultados do teste de estresse do *UIoT Middleware* nos *gateways* para envio de dados com intervalo de 1s.

Pode-se concluir que o *UIoT Middleware* ainda possui problemas relacionados a armazenamento e repasse de dados quando está rodando em um dispositivo com menos recursos, como o Raspberry Pi 3, principalmente quando o fluxo de dados é superior a uma requisição por segundo.

4.2 Cenário com falha no *Gateway* Mestre

Caso o GW-02, em estado de *backup*, venha a perder conectividade com a rede, o GW-01 passa a atuar de maneira independente, realizando ele mesmo o processamento dos novos dados recebidos, ainda como *Gateway* Mestre, possuindo o VIP vinculado à sua interface de rede sem fio. Como pode ser visto na figura 4.11, apenas o GW-01 está acessível agora.

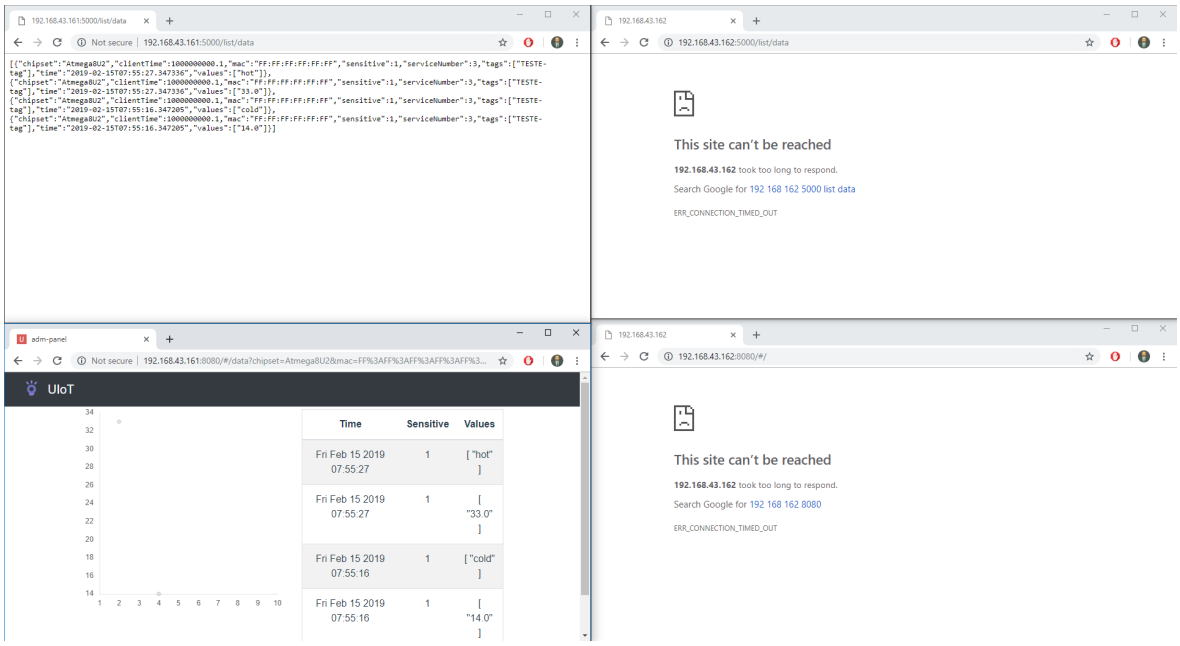

Figura 4.11: *Gateway Backup* (GW-02) indisponível na rede, mas *Gateway* Mestre (GW-01) ainda comunicável.

Ao enviar uma nova informação sobre a temperatura do ambiente, os dados são recebidos pelo GW-01, são repassados para o *UIoT Middleware* local, para o *UIoT Middleware* hospedado na nuvem e o processamento ocorre localmente. As novas informações, tanto as originais quanto as processadas, são exibidas diretamente pelo *UIoT Middleware* do GW-01.

Quando o GW-02 volta a ter conectividade com a rede, o cenário sem falhas é refeito, com o *Gateway Backup* cuidando da parte de processamento e visualização de dados.

4.3 Cenário com falha no *Gateway Backup*

Caso o GW-01 perca conectividade com a rede, a situação inversa ocorre. O GW-02 assume o VIP, como mostrado na figura 4.13, e atua no papel de *Gateway* Mestre, cuidando tanto da parte de comunicação quanto da parte de processamento e exibição de dados, até que o GW-01 volte a se comunicar na rede.

Como agora o GW-01 é o *gateway* que não está acessível, todos os seus serviços se encontram indisponíveis, ao contrário dos serviços do GW-02, como mostrado na figura 4.14 abaixo.

Quando uma nova informação é enviada ao VIP, o GW-02 recebe os novos dados, repassa-os para o seu *UIoT Middleware* local, para o *UIoT Middleware* na nuvem e realiza o

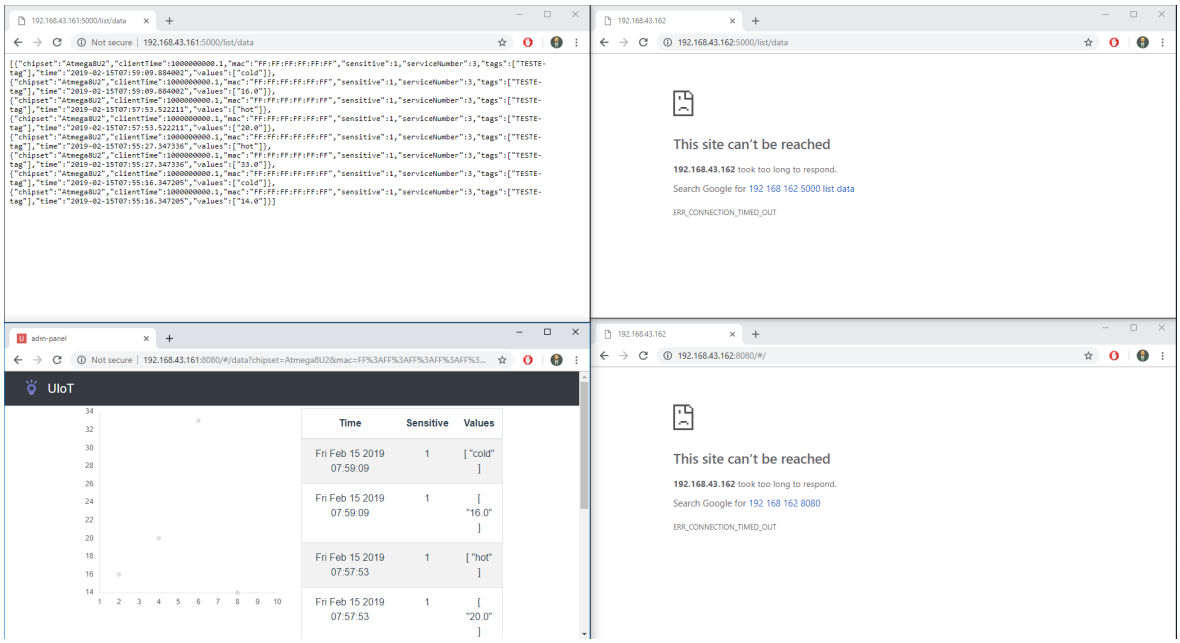

Figura 4.12: Novas informações (originais e processadas) adicionadas ao banco de dados.

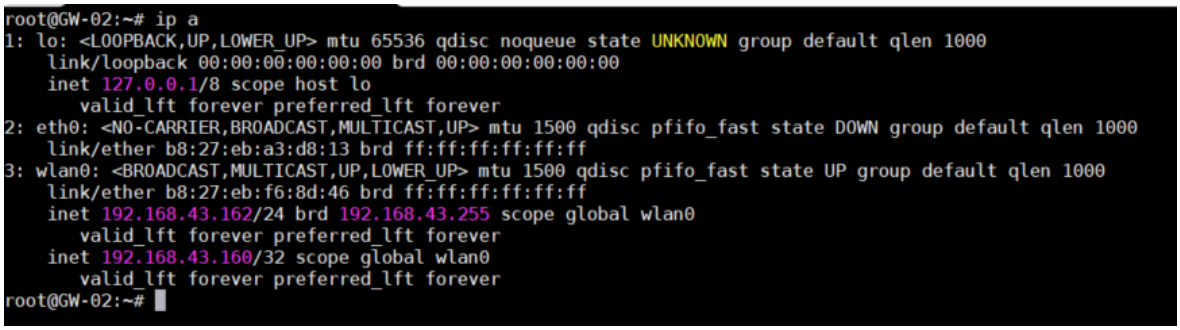

Figura 4.13: Endereços IP do GW-02 após a desconexão do GW-01.

processamento localmente. Os dados são armazenados e exibidos pelo *UIoT Middleware* do GW-01.

A figura 4.16 abaixo mostra os dados enviados para o *UIoT Middleware* em execução no Heroku, uma plataforma para aplicações na nuvem. Os dados processados não são enviados para a nuvem, já que os dados numéricos fazem mais sentido para a análise em grandes quantidades.

Ao ligar o GW-01, ele inicialmente entra em estado de *backup* (LED na cor roxa, na figura 4.17), até que recebe anúncios do atual mestre, o GW-02, que possui uma prioridade menor que o GW-01. Por conta disso, o GW-01 força que uma nova eleição seja feita, a fim de mudar para o estado de mestre. Quando os papéis se invertem e os dispositivos retornam para o cenário inicial sem falhas, o LED do GW-01 fica verde, indicando que ele agora está no estado de mestre, enquanto o LED do GW-02 muda para a cor roxa, indicando que ele é o *Gateway Backup* novamente (figura 4.18.

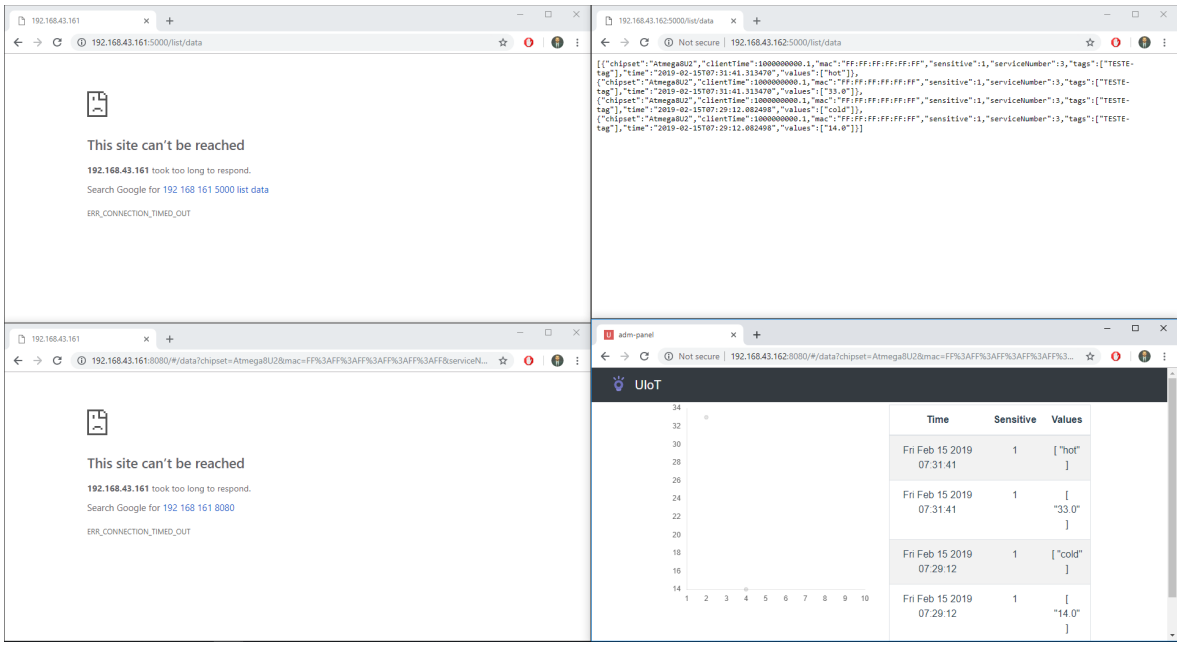

Figura 4.14: Situação do banco de dados dos *gateways* após a queda do GW-01.

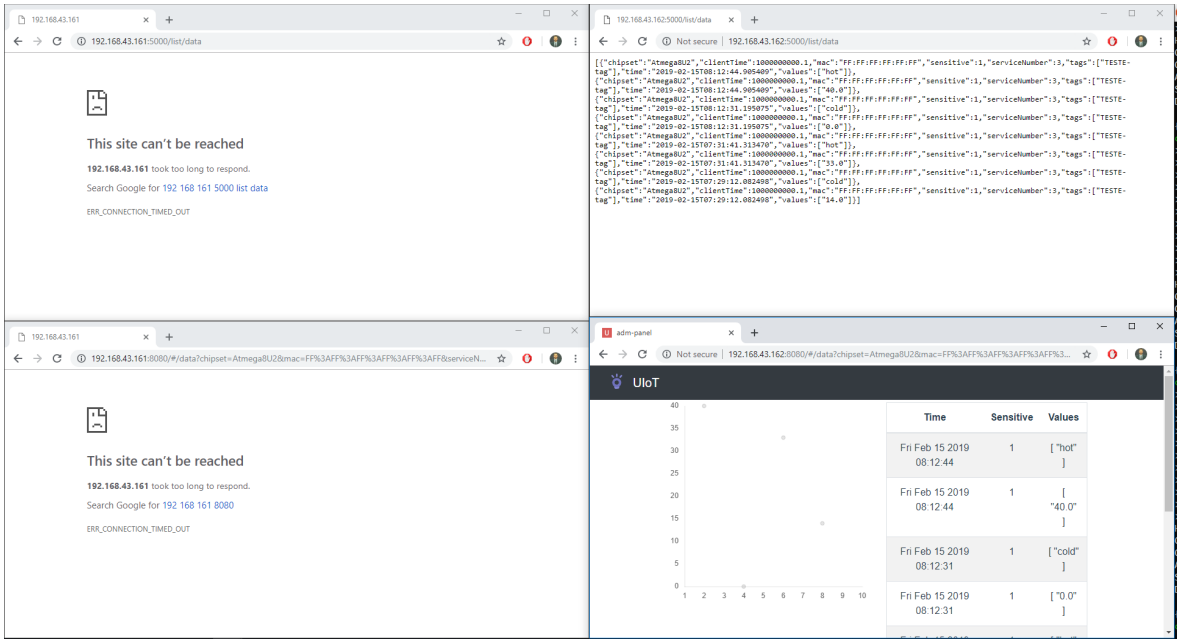

Figura 4.15: Dados originais e processados estão no próprio GW-02.

Figura 4.16: Dados enviados de dispositivos locais e repassados para o *UIoT Middleware* hospedado na nuvem.

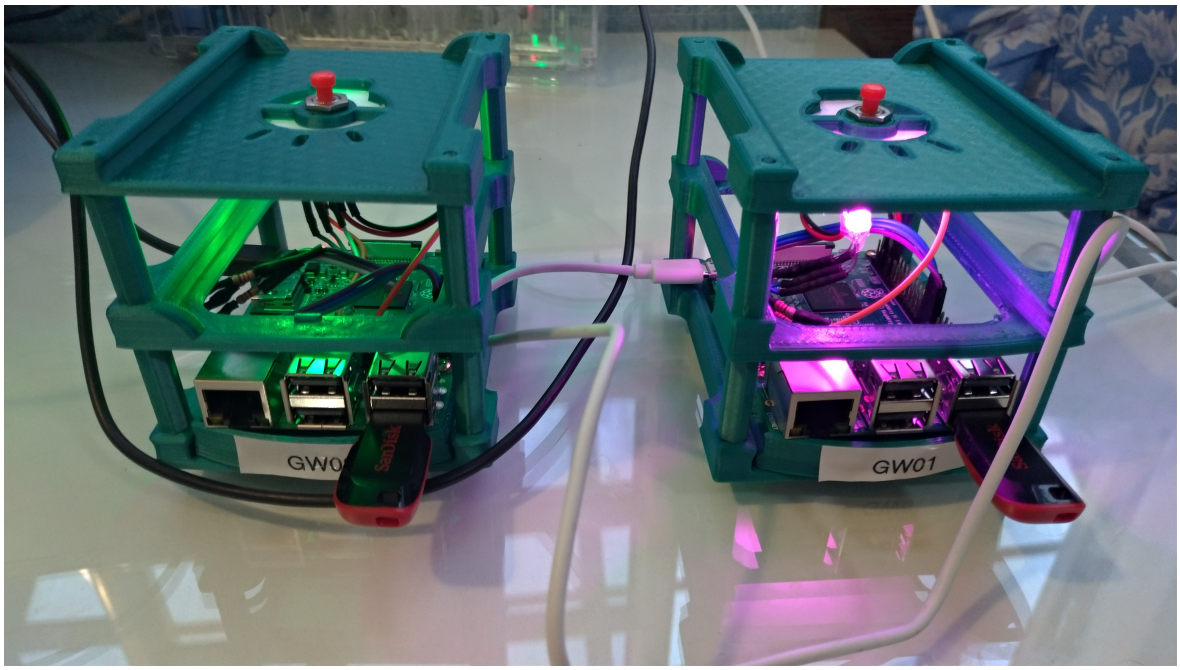

Figura 4.17: Indicação de LEDs quando o GW-01 é o *backup* (roxo) e o GW-02 é o mestre (verde).

Figura 4.18: Logo após a eleição, o cenário sem falhas é recomposto, com o GW-01 como mestre e com LED verde e o GW-02 como *backup* e com LED roxo.

Capítulo 5

Conclusão

O trabalho abordou conceitos recentes da área, como Internet das Coisas, *Cloud*, *Fog* e *Mist Computing*, além de explorar bastante as ideias de *gateway*, redundância, alta disponibilidade e latência. *Fog Computing* e Internet das Coisas tendem a ganhar espaço na indústria e no mercado de trabalho, por isso realizar um trabalho nesse assunto pode gerar outros benefícios ao pesquisador além do conhecimento adquirido.

A proposta apresentada no presente trabalho foi um *gateway* redundante IoT com uso de *Fog Computing*, que proveria alta disponibilidade ao *gateway* da rede IoT, garantindo menos interrupções na transmissão de dados dos dispositivos IoT presentes no ambiente, ao mesmo tempo em que oferece a experiência da *Cloud Computing* dentro da própria rede local, permitindo que dados sejam armazenados, processados e exibidos aos usuários de maneira simples.

O protótipo tem um design bonito e suas funções são úteis para uma rede IoT, mesmo que não seja um ambiente crítico, até porque o Raspberry Pi 3 não oferece um equipamento tão robusto para ser utilizado em redes maiores. Nos cenários de teste, os protótipos cumpriram seus papéis, apesar de ainda haver várias melhorias a serem feitas para garantir maior estabilidade à proposta.

Há várias pesquisas em torno de *gateway* IoT, como os artigos (ZHU et al., 2010) e (KRYLOVSKIY, 2015), mas não foi possível encontrar artigos específicos sobre redundância de *gateways* IoT. Há informações na Internet sobre esse tópico (BHARDWAJ, 2017), mas nada em desenvolvimento. É bem provável que haja soluções proprietárias, mas não parecem ser residenciais pela dificuldade de achá-las na Internet.

Por isso, o protótipo desenvolvido nesse trabalho teria como diferencial ser uma solução mais simples, barata e focada em automação residencial, onde também não receberia um fluxo tão grande de dados.

Por fim, há vários artigos abordando *Fog Computing* e IoT, como (AAZAM; HUH, 2014), o que quase sempre leva os autores a propor ou desenvolver um *smart gateway* ou um *gateway* IoT com algum tipo de *middleware* embutido. Como o *UIoT Middleware* ainda está em desenvolvimento, foi utilizado nesse trabalho apenas para exemplificar certos benefícios da *Fog Computing* em uma rede IoT.

5.1 Trabalhos Futuros

Como trabalhos futuros, há uma vasta variedade de assuntos a se explorar e tecnologias a desenvolver, por ser uma área abrangente, nova e em crescimento.

Mais especificamente relacionado à redundância, há a possibilidade criar um *cluster* de *gateway* IoT com mais de 2 *gateways*, provendo balanceamento de carga tanto na parte de comunicação quanto na de processamento de dados. Também é possível comparar os FHRP e validar se vale a pena utilizar CARP ou outro protocolo no lugar do VRRP.

Em relação ao *UIoT Middleware*, é possível desenvolvê-lo para suportar mais protocolos, principalmente *Bluetooth* e *ZigBee*, mantendo a redundância. Esse trabalho teria a complexidade de ter de realizar a mudança de conexão entre os dispositivos IoT e os *gateways* toda vez que o *Gateway* Mestre do VRRP fosse substituído. Também seria possível desenvolvê-lo para prover melhor comunicação com a nuvem, enviando os dados condensados, para economizar banda, além de poder controlar melhor os atuadores da rede local.

Por fim, também seria possível desenvolver o protótipo originado nesse trabalho. Há várias melhorias a serem feitas, principalmente no que se diz respeito à estabilidade do sistema operacional, ao armazenamento de dados e à execução dos programas. Também seria ideal generalizar sua aplicação, de modo que ele se adaptasse a qualquer rede, a fim de torná-lo um produto de fato, que pode ser utilizado em qualquer rede por qualquer pessoa. A implementação de um servidor DNS para IoT nele também pode ser bastante útil.

Bibliografia

GARTNER, INC. **Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016**. Fev. 2017. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>. Acesso em: 6 jan. 2019.

IHS MARKIT. **Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says**. Out. 2017. Disponível em: <https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says>. Acesso em: 6 jan. 2019.

STACK, Tim. **Data Center Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How?** Fev. 2018. Disponível em: <https://blogs.cisco.com/datacenter/internet-of-things-iot-data-continues-to-explode-exponentially-who-is-using-that-data-and-how>. Acesso em: 6 jan. 2019.

STATISTA. **Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)**. Nov. 2016. Disponível em: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>. Acesso em: 6 jan. 2019.

PANDIT, Puneet. **The Impact of Machine Data Analytics, Artificial Intelligence and Machine Learning on Healthcare Technology**. Set. 2018. Disponível em: <https://www.glassbeam.com/blog/impact-machine-data-analytics-artificial-intelligence-and-machine-learning-healthcare>. Acesso em: 14 fev. 2019.

DQE COMMUNICATIONS. **IoT and its impact on bandwidth**. Mar. 2018. Disponível em: <https://www.dqecom.com/resources/tech-talk/iot-impact-bandwidth/>. Acesso em: 7 jan. 2019.

EMBITEL. **What is an IoT Gateway Device and Why is it so Important for the Success of IoT Projects?** Out. 2017. Disponível em: <https://www.embitel.com/blog/embedded-blog/what-is-an-iot-gateway-device-and-why-is-it-so-important-for-the-success-of-iot-projects>. Acesso em: 7 jan. 2019.

GARTNER, INC. **The Cost of Downtime**. Jul. 2014. Disponível em: <https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/>. Acesso em: 14 jan. 2019.

NATIONAL INSTRUMENTS CORPORATION. **Redundant System Basic Concepts**. Jan. 2008. Disponível em: <http://www.ni.com/white-paper/6874/en/>. Acesso em: 14 jan. 2019.

NWN. **Why Fog is Better Than Cloud Cover in IoT**. Jun. 2018. Disponível em: <https://www.nwnit.com/blog/2018/06/25/why-fog-is-better-than-cloud-cover-in-iot/>. Acesso em: 7 fev. 2019.

YI, Shanhe; LI, Cheng; LI, Qun. A survey of fog computing: concepts, applications and issues, p. 37–42, 2015.

CHIANG, Mung; ZHANG, Tao. Fog and IoT: An Overview of Research Opportunities. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 6, p. 854–864, dez. 2016. ISSN 2327-4662. DOI: 10.1109/JIOT.2016.2584538.

TANENBAUM, Andrew S.; WETHERALL, David J. **Computer Networks**. 5th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0132126958.

KUROSE, James F.; ROSS, Keith W. **Redes de computadores e a internet: uma abordagem top-down**. [S.l.]: Pearson, 2013.

FREEMAN, Roger L. **Fundamentals of Telecommunication**. 2nd. [S.l.]: IEEE Press, 2005. ISBN 9780471720942.

ZHU, Q. et al. IOT Gateway: Bridging Wireless Sensor Networks into Internet of Things, p. 347–352, dez. 2010. DOI: 10.1109/EUC.2010.58.

AL-FUQAHA, A. et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2347–2376, 2015. ISSN 1553-877X. DOI: 10.1109/COMST.2015.2444095.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks, p. 791–798, jan. 2008. DOI: 10.1109/COMSWA.2008.4554519.

AAZAM, M.; HUH, E. Fog Computing and Smart Gateway Based Communication for Cloud of Things, p. 464–470, ago. 2014. DOI: 10.1109/FiCloud.2014.83.

THE MATHWORKS, INC. **Learn More About ThingSpeak**. Disponível em: <https://thingspeak.com/pages/learn_more>. Acesso em: 3 fev. 2019.

WI-FI ALLIANCE. **Discover Wi-Fi**. Jan. 2018. Disponível em: <https://www.wi-fi.org/discover-wi-fi>. Acesso em: 3 fev. 2019.

IETF. **Request for Comments: 2616 - Hypertext Transfer Protocol – HTTP/1.1**. Jun. 1999. Disponível em: <https://www.ietf.org/rfc/rfc2616.txt>. Acesso em: 4 fev. 2019.

MQTT.ORG. **Frequently Asked Questions**. Jan. 2018. Disponível em: <http://mqtt.org/faq>. Acesso em: 4 fev. 2019.

THE HIVEMQ TEAM. **MQTT Essentials Part 2: Publish and Subscribe**. Jan. 2015. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/>. Acesso em: 4 fev. 2019.

FIWARELAB. **Apresentação do protocolo MQTT como alternativa para comunicação IoT**. Jul. 2016. Disponível em: <http://fiwarelabsp.org/2016/07/apresentacao-do-protocolo-mqtt/>. Acesso em: 4 fev. 2019.

THE HIVEMQ TEAM. **MQTT Essentials Part 5: MQTT Topics and Best Practices**. Fev. 2015. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>. Acesso em: 4 fev. 2019.

DAVIDSON, Robert et al. **Getting Started with Bluetooth Low Energy by Robert Davidson, Akiba, Carles Cufí, Kevin Townsend**. Disponível em: <https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch01.html>. Acesso em: 4 fev. 2019.

ZIGBEE ALLIANCE. **Standards: ZigBee Specification**. Dez. 2014. Disponível em: <https://www.zigbee.org/download/standards-zigbee-specification/>. Acesso em: 4 fev. 2019.

CHARARA, Sophie. **Zigbee vs Z-Wave: Two big smart home standards explored**. Dez. 2018. Disponível em: <https://www.the-ambient.com/guides/zigbee-vs-z-wave-298>. Acesso em: 4 fev. 2019.

SAWH, Michael. **The best smart clothing: From biometric shirts to contactless payment jackets**. Abr. 2018. Disponível em: <https://www.wareable.com/smart-clothing/best-smart-clothing>. Acesso em: 16 fev. 2019.

WEISER, Mark. The Computer for the 21st Century. **Scientific American**, v. 265, n. 3, p. 66–75, set. 1991. Acesso em: 3 fev. 2019.

WELCH, Chris. **Google just gave a stunning demo of Assistant making an actual phone call**. Maio 2018. Disponível em: <https://www.theverge.com/2018/5/8/17332070/google-assistant-makes-phone-call-demo-duplex-io-2018>. Acesso em: 16 fev. 2019.

MAHESA, Raka. **How cloud, fog, and mist computing can work together**. Mar. 2018. Disponível em: <https://developer.ibm.com/articles/how-cloud-fog-and-mist-computing-can-work-together/>. Acesso em: 3 fev. 2019.

ARCITURA EDUCATION INC. **WhatIsCloud.com - Goals and Benefits**. Disponível em: <http://whatiscloud.com/goals_and_benefits/index>. Acesso em: 16 fev. 2019.

LINTHICUM, David. **Edge computing vs. fog computing: Definitions and enterprise uses**. Disponível em: <https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html>. Acesso em: 1 fev. 2019.

MAIER, Martin; SAME, Mohammad Hossein. **How cloud, fog, and mist computing can work together**. Mar. 2018. Disponível em: <http://www.zeitgeistlab.ca/doc/context_and_self_awareness_in_fog_and_mist_computing.html>. Acesso em: 1 fev. 2019.

MARKAKIS, Evangelos K. et al. EXEGESIS: Extreme edge resource harvesting for a virtualized fog environment. English. **IEEE Communications Magazine**, Institute of Electrical e Electronics Engineers Inc., v. 55, n. 7, p. 173–179, jul. 2017. ISSN 0163-6804. DOI: 10.1109/MCOM.2017.1600730.

OPSERVICES. **MTTR AND MTBF, WHAT ARE THEY AND WHAT ARE THEIR DIFFERENCES?** Ago. 2015. Disponível em: <https://www.opservices.com/mttr-and-mtbf/>. Acesso em: 18 jan. 2019.

SWANSON, Laura. Linking maintenance strategies to performance. **International Journal of Production Economics**, Elsevier BV, v. 70, n. 3, p. 237–244, abr. 2001. DOI: 10.1016/s0925-5273(00)00067-0.

ORACLE CORPORATION. **Availability and Single Points of Failure**. 2010. Disponível em: <https://docs.oracle.com/cd/E19424-01/820-4806/fjdch/index.html>. Acesso em: 18 jan. 2019.

NGINX. **What Is Load Balancing?** Disponível em: <https://www.nginx.com/resources/glossary/load-balancing/>. Acesso em: 18 jan. 2019.

CISCO. **Catalyst 3560 Software Configuration Guide, Release 12.2(52)SE - Chapter: Configuring HSRP**. Abr. 2017. Disponível em: <https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3560/software/release/12-2_52_se/configuration/guide/3560scg/swhsrp.html>. Acesso em: 9 jan. 2019.

IETF. **Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6**. Mar. 2010. Disponível em: <https://tools.ietf.org/html/rfc5798>. Acesso em: 19 jan. 2019.

CISCO. **GLBP - Gateway Load Balancing Protocol**. Disponível em: <https://www.cisco.com/en/US/docs/ios/12_2t/12_2t15/feature/guide/ft_glbp.html>. Acesso em: 19 jan. 2019.

OPENBSD. **PF - Firewall Redundancy (CARP and pfsync)**. Disponível em: <https://www.openbsd.org/faq/pf/carp.html>. Acesso em: 19 jan. 2019.

BROWN, Eric. **Raspberry Pi 3 has 64-bit CPU, but 32-bit Raspbian OS (for now)**. Mar. 2016. Disponível em: <http://linuxgizmos.com/raspberry-pi-3-has-a-64-bit-cpu-but-a-32-bit-raspbian-os/>. Acesso em: 23 jan. 2019.

DEBIAN. **RaspberryPi3**. Fev. 2019. Disponível em: <https://wiki.debian.org/RaspberryPi3>. Acesso em: 23 jan. 2019.

UIOT. **UIoT**. Fev. 2019. Disponível em: <https://github.com/uiot>. Acesso em: 24 jan. 2019.

FERREIRA, Hiro Gabriel Cerqueira et al. RAISe: REST API Approach for IoT Services.

KEEPALIVED. **Keepalived for Linux - Keepalived Configuration Manual Page**. Nov. 2018. Disponível em: <http://www.keepalived.org/manpage.html>. Acesso em: 24 jan. 2019.

WENSONG. **What is virtual server?** Fev. 2011. Disponível em: <http://www.linux-vs.org/whatis.html>. Acesso em: 24 jan. 2019.

_____. **IPVS**. Mar. 2016. Disponível em: <http://www.linuxvirtualserver.org/software/ipvs.html>. Acesso em: 24 jan. 2019.

IETF. **Bidirectional Forwarding Detection (BFD)**. Jun. 2010. Disponível em: <https://tools.ietf.org/html/rfc5880>. Acesso em: 24 jan. 2019.

MONGODB. **What is MongoDB?** Jan. 2019. Disponível em: <https://www.mongodb.com/what-is-mongodb>. Acesso em: 24 jan. 2019.

_____. **32-bit limitations**. Jul. 2009. Disponível em: <https://www.mongodb.com/blog/post/32-bit-limitations>. Acesso em: 24 jan. 2019.

KRYLOVSKIY, A. Internet of Things gateways meet linux containers: Performance evaluation and discussion, p. 222–227, dez. 2015. DOI: 10.1109/WF-IoT.2015.7389056.

BHARDWAJ, MOHIT. **How IoT Gateway Clustering Ensures Reliability and High Availability**. Ago. 2017. Disponível em: <https://www.einfochips.com/blog/iot-gateway-architecture-clustering-ensures-reliability/>. Acesso em: 17 fev. 2019.

ANEXOS

ANEXO I

Configurando os *Gateways* IoT

Abaixo estão documentados todas as modificações e os *scripts* criados para reproduzir a proposta apresentada. As instruções abaixo devem ser realizadas duas vezes, uma em cada dispositivo, já que eles compõem um *cluster*.

I.1 Realizando flash da imagem do *Debian Buster*

Para realizar o *download* da imagem do *Debian Buster* 64-bit para Raspberry Pi 3, basta seguir para o caminho:

https://people.debian.org/ gwolf/raspberrypi3/20190206/20190206-raspberry-pi-3-buster-PREVIEW.img.xz

Depois é necessário realizar o *flash* dessa imagem em um cartão micro SD, de preferência com 32GB ou mais. Para isso, foi utilizado o *software* para *Windows* chamado *balenaEtcher*, que pode ser encontrado no seguinte *link*:

https://www.balena.io/etcher/

Após realizar o *flash* da imagem do sistema operacional, basta inserir o cartão micro-SD no Raspberry Pi 3 e ligá-lo. O usuário do sistema é "*root*"e a senha é "*raspberry*".

I.2 Preparação do Raspberry Pi 3 Model B: Wi-Fi e Atualizações

Para conectar o Raspberry Pi 3 a uma rede sem fio, é necessário inserir os seguintes comandos:

```
$ wpa_passphrase SSID_DA_REDE SENHA_DA_REDE > /etc/
   ↪ wpa_supplicant/wpa_supplicant.conf
$ ip link set wlan0 down
$ ip link set wlan0 up
$ wpa_supplicant -B -iwlan0 -c /etc/wpa_supplicant/
   ↪ wpa_supplicant.conf
$ dhclient wlan0
```

Quando tiver conexão com a Internet, é possível realizar a atualização do sistema.

```
$ apt update -y
$ apt upgrade -y
$ apt dist-upgrade -y
$ reboot now
```

Para definir um endereço IP estático para o dispositivo, é necessário editar o arquivo "/etc/network/interfaces"e inserir as informações referentes à rede a qual se deseja conectar:

```
allow-hotplug wlan0
iface wlan0 inet static
      wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
      address 192.168.43.161 #192.168.43.162 para o GW-02
      netmask 255.255.255.0
      gateway 192.168.43.1
```

Reinicie o sistema mais uma vez com o comando "*reboot now*". Com o sistema agora atualizado e conectado à rede, é possível instalar os programas necessários para configurar o *gateway* IoT. O seguinte comando deve ser inserido:

```
$ apt install -y vim net-tools mongodb git python3-pip
   ↪ python3-venv keepalived npm
```

I.3 Configurações para o MongoDB

O *MongoDB* exige muito uso de memória quando se realiza *dump* e *restore*, o que será feito para sincronizar os bancos de dados, então é necessário disponibilizar um espaço de *swap* para que não haja problema de falta de memória, já que o Raspberry Pi 3 Model B possui apenas 1 GB de memória RAM. O *swap* pode ser criado inserindo os seguintes comandos:

```
$ touch /tmp/theswap
$ chmod 600 /tmp/theswap
$ dd if=/dev/zero of=/tmp/theswap bs=1M count=2048
$ mkswap /tmp/theswap
$ swapon /tmp/theswap
```

Para utilizar o pendrive como espaço de armazenamento para *MongoDB*, é necessário formatá-lo como Ext2. É necessário adicionar um comando ao *Crontab* para montar o pendrive num caminho específico a cada inicialização do sistema. Isso será feito mais tarde. Agora é necessário montar o pendrive utilizando o comando abaixo na linha de comando e criar uma pasta chamada "mongodb"na pasta "/media"e mudar o grupo e o dono da pasta também para "mongodb", para permitir que o *MongoDB* escreva no pendrive:

```
$ mount -t ext2 /dev/sda1 /media
$ mkdir /media/mongodb
$ chown -R mongodb:mongodb /media/mongodb
```

Por fim, basta editar o arquivo "/etc/mongodb.conf"para os seguintes parâmetros:

```
dbpath=/media/mongodb
bind_ip = 0.0.0.0
port = 27017
```

O parâmetro "dbpath"indica o caminho onde o *MongoDB* salvará os dados dos bancos de dados. Os parâmetros "bind_ip"e "port"tornam possível o acesso remoto ao *MongoDB* para realizar a sincronia de banco de dados. Para iniciar e verificar o estado do serviço do *MongoDB*, é necessário inserir os seguintes comandos:

```
$ service mongodb restart
$ service mongodb status
```

I.4 Instalação do UIoT Middleware

Com o Git, realize o clone dos repositórios dos *middlewares UIoT Gateway*, *UIoT DIMS* e *UIoT UIMS*:

```
$ git clone https://github.com/uiot/dims.git
$ git clone https://github.com/uiot/gateway.git
$ git clone https://github.com/uiot/publisher.git
$ git clone https://github.com/uiot/uims.git
```

Crie um ambiente virtual (*venv*) e crie a pasta de *log* do *UIoT Gateway*:

```
$ python3 -m venv venv
$ mkdir /var/log/uiot
$ chown $USER:$USER /var/log/uiot
```

Ative o ambiente virtual e realize a instalação do *UIoT Gateway*:

```
$ source venv/bin/activate
$ cd /root/gateway/
$ git checkout egg
$ pip install -e .
```

Realize a instalação do *UIoT DIMS* sem sair do ambiente virtual até o término da instalação:

```
$ cd /root/dims/
$ pip install -e .
```

Ainda no ambiente virtual, instale o *UIoT Publisher*:

```
$ cd /root/publisher
$ git checkout mvpegg
$ pip install -e .
$ deactivate
```

Por fim, há a instalação do *UIoT UIMS* fora do ambiente virtual:

```
$ cd /root/uims/
$ git checkout adm-data-visualization
$ npm install
```

Execute o comando abaixo para atualizar a URL do *UIoT DIMS* e do *gateway* que ficará como *backup*:

```
$ echo "export␣DIMS_URL=http://localhost:5000/" > /etc/
  ↪ environment
$ echo "export␣BACKUP_IP=192.168.43.162" >> /etc/environment
```

I.5 Configuração do *Keepalived*

Para que o *cluster* VRRP funcione, edite o arquivo "/etc/keepalived/keepalived.conf"do GW-01 para ficar da seguinte maneira:

```
! Configuration File for keepalived

vrrp_instance VI_1 {
   state MASTER
   interface wlan0
   virtual_router_id 100
   priority 100
   advert_int 1

   virtual_ipaddress {
      192.168.43.160
   }

   notify /root/current_state.sh
}
```

Agora edite o arquivo "/etc/keepalived/keepalived.conf"do GW-02 para ficar da seguinte maneira e com uma prioridade menor:

```
! Configuration File for keepalived

vrrp_instance VI_1 {
   state BACKUP
   interface wlan0
   virtual_router_id 100
   priority 10
   advert_int 1

   virtual_ipaddress {
      192.168.43.160
   }

   notify /root/current_state.sh
}
```

Depois, inicialize o *Keepalived* e verifique seu estado utilizando os seguintes comandos:

```
$ service keepalived start
$ service keepalived status
$ ipconfig
```

Após o comando "*ipconfig*", você deverá ver o IP virtual (VIP) na interface indicada em algum dos dois dispositivos. O que tiver o VIP será o *Gateway* Mestre. Se ele for desligado, o *Gateway Backup* assumirá o VIP e se tornará o *Gateway* Mestre. Verifique se isso ocorre antes de prosseguir.

I.6 Scripts no Crontab e em Python

Na home do *root*, crie o programa "/root/control_led.py"em Python:

```
cores={'vermelho':'011','verde':'110','amarelo':'010','azul':
   ↪ '101','branco':'000','ciano':'100','roxo':'001','off':'
   ↪ 111'}
import os
import sys

def inicializa():
      os.system('echo 471 > /sys/class/gpio/export')
      os.system('echo 464 > /sys/class/gpio/export')
      os.system('echo 463 > /sys/class/gpio/export')
      os.system('echo "out" > /sys/class/gpio/gpio471/
         ↪ direction')
      os.system('echo "out" > /sys/class/gpio/gpio464/
         ↪ direction')
      os.system('echo "out" > /sys/class/gpio/gpio463/
         ↪ direction')

def controla_led(rgb):
      os.system('echo '+ str(rgb[0]) +' > /sys/class/gpio/
         ↪ gpio471/value')
      os.system('echo '+ str(rgb[1]) +' > /sys/class/gpio/
         ↪ gpio464/value')
      os.system('echo '+ str(rgb[2]) +' > /sys/class/gpio/
         ↪ gpio463/value')

try:
      inicializa()
except:
      desligado()

param = sys.argv[1]
if param == 'MASTER':
      controla_led(cores['verde'])
elif param == 'BACKUP':
      controla_led(cores['roxo'])
```

Ainda na home do *root*, crie o arquivo "/root/current_state.sh", que notificará os *scripts* sobre o estado do dispositivo (*MASTER*, *BACKUP* ou outro):

```
#!/bin/bash

TYPE=$1
NAME=$2
STATE=$3
```

```
echo $STATE > /root/state.txt
python /root/control_led.py $STATE
```

Último script na home do root é o "/root/shutdown.py", que servirá para desligar o Raspberry Pi pelo botão do conjunto:

```
import subprocess
import os

def inicializa():
  os.system('echo 461 > /sys/class/gpio/export')
  os.system('echo "in" > /sys/class/gpio/gpio461/direction')

inicializa()
while True:
  file = open('/sys/class/gpio/gpio461/value','r')
  released = file.read(1)
  file.close
  if str(released) == '0':
    subprocess.call(['shutdown', '-h', 'now'], shell=False)
```

Utilize a pasta "/etc/cron.d/"para manter os scripts que serão adicionados ao *Crontab*. Dentro dessa pasta, crie um arquivo "/etc/cron.d/middleware_init.sh"com o seguinte conteúdo:

```
#!/bin/bash

sleep 120

. /root/venv/bin/activate
dims &
sleep 30
gateway -p http -f
sleep 15
publisher &> /dev/null &
sleep 15
deactivate

cd /root/uims
npm run serve &
```

Crie também dentro da mesma pasta o *script* "/etc/cron.d/publisher.sh":

```
#!/bin/bash

. /root/venv/bin/activate

#UIOT=0 # MASTER ALONE
UIOT=1 # MASTER # Use essa opcao como padrao para o GW-01
#UIOT=2 # BACKUP # Use essa opcao como padrao para o GW-02
```

```
ping -c 2 $BACKUP_IP &> /dev/null

if [ $? != 0 ]; then
   $UIOT=0
fi

publisher &

deactivate
```

Adicione as permissões necessárias para executar esses *scripts* da seguinte maneira:

```
$ chmod +x /root/current_state.sh
$ chmod +x /etc/cron.d/dims_uims.sh
$ chmod +x /etc/cron.d/gateway.sh
$ chmod +x /etc/cron.d/publisher.sh
```

Por fim, use o comando "crontab -e"para abrir o *script* do *Crontab* e adicione as seguintes linhas ao final:

```
@reboot mount -t ext2 /dev/sda1 /media
@reboot /etc/cron.d/middleware_init.sh
@reboot /etc/cron.d/publisher.sh
@reboot python /root/shutdown.py &
@reboot python /root/control_led.py "$(cat␣/root/state.txt)"
```

A primeira linha monta o pendrive na pasta "/media"na inicialização do sistema operacional. A segunda linha coloca o *UIoT DIMS* e o *UIoT UIMS* em execução na inicialização do sistema. A terceira linha inicializa o *UIoT Gateway* um minuto após a inicialização do sistema.

A quarta linha inicializa o *Publisher* junto com o *boot* do sistema. A quinta linha permite que o dispositivo seja desligado através de um botão presente no compartimento onde a placa do Raspberry Pi fica apoiada. A sexta linha realiza o controle do LED de acordo com o estado do *gateway* em relação ao VRRP (se for *backup*, o LED será roxo; se for mestre, o LED será verde; se o LED for branco, está em algum estado desconhecido; e se está apagado, o dispositivo está desligado).

I.7 Envios de dados de teste

Para os comandos abaixo, é necessário utilizar a ferramenta *curl*.

Para adicionar um cliente:

```
curl -i --request POST \
   --url http://192.168.43.160:8000/client \
   --header 'content-type: application/json' \
   --data '{
      "clientTime": 1000000000.1111,
      "tags": [ "TESTE-tag" ],
      "name": "Arduino-Uno-Teste",
      "chipset": "Atmega8U2",
      "mac": "FF:FF:FF:FF:FF:FF",
      "serial": "C210",
      "processor": "ATmega328",
      "channel": "Ethernet",
      "location": "-15.7757876:-48.077829"
   }'
```

Para adicionar um serviço:

```
curl -i --request POST \
   --url http://192.168.43.160:8000/service \
   --header 'content-type: application/json' \
   --data '{
      "clientTime": 1000000000.1,
      "tags": [ "TESTE-tag" ],
      "number": 3,
      "chipset": "Atmega8U2",
      "mac": "FF:FF:FF:FF:FF:FF",
      "name": "Get␣temp",
      "parameter": "temperature",
      "unit": "oC",
      "numeric": 1
   }'
```

Para adicionar um dado:

```
curl -i --request POST \
   --url http://192.168.43.160:8000/data \
   --header 'content-type: application/json' \
   --data '{
      "clientTime": 1000000000.1,
      "tags": [ "TESTE-tag" ],
      "sensitive": 1,
      "chipset": "Atmega8U2",
      "mac": "FF:FF:FF:FF:FF:FF",
      "serviceNumber": 3,
      "values": [ "40.0" ]
   }'
```

O *script* em *bash* utilizado para testar o *UIoT Middleware* foi o seguinte:

```
#!/bin/bash

for i in {1..100}
do
   echo "Valor: " $i

   curl --request POST \
      --url http://192.168.43.160:8000/data \
      --header 'content-type: application/json' \
      --data '{
         "clientTime": 1000000000.1,
         "tags": [ "TESTE-tag" ],
         "sensitive": 1,
         "chipset": "Atmega8U2",
         "mac": "FF:FF:FF:FF:FF:FF",
         "serviceNumber": 3,
         "values": [ '"${i}"' ]
      }'

   sleep 1
done
```