PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIOT

FÁBIO LÚCIO LOPES DE MENDONÇA

TESE DE DOUTORADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIOT

FÁBIO LÚCIO LOPES DE MENDONÇA

Orientador: PROF. DR. RAFAEL TIMÓTEO DE SOUSA JÚNIOR, ENE/UNB

TESE DE DOUTORADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO PPGEE.TD - 150/2019
BRASÍLIA-DF, 16 DE JULHO DE 2019.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

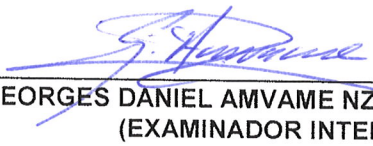
PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIOT

FÁBIO LÚCIO LOPES DE MENDONÇA

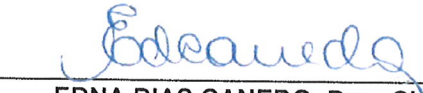
TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.

APROVADA POR:

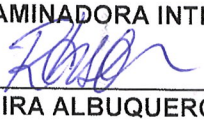
RAFAEL TIMÓTEO DE SOUSA JÚNIOR, Dr., ENE/UNB
(PRESIDENTE DA COMISSÃO)



GEORGES DANIEL AMVAME NZE, Dr., ENE/UNB
(EXAMINADOR INTERNO)



EDNA DIAS CANEDO, Dra., CIC/UNB
(EXAMINADORA INTERNA)



ROBSON DE OLIVEIRA ALBUQUERQUE, Dr., CEPESC/PR
(EXAMINADOR EXTERNO)

Brasília, 16 de julho de 2019.

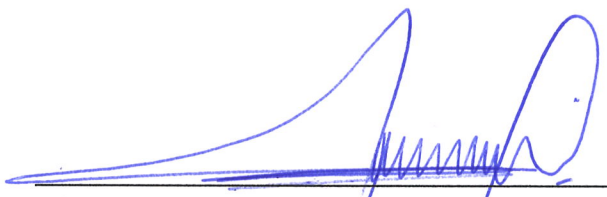
FICHA CATALOGRÁFICA
FÁBIO LÚCIO LOPES DE MENDONÇA
PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIoT
2019xv, 140p., 201x297 mm
(ENE/FT/UnB, Doutor, Engenharia Elétrica, 2019)
Tese de Doutorado - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

FÁBIO LÚCIO LOPES DE MENDONÇA (2019) PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIoT. Tese de Doutorado em Engenharia Elétrica, Publicação PPGEE.TD 150/2019, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 140p.

CESSÃO DE DIREITOS

AUTOR: FÁBIO LÚCIO LOPES DE MENDONÇA
TÍTULO: PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIoT.
GRAU: Doutor ANO: 2019



FÁBIO LÚCIO LOPES DE MENDONÇA
SHVP Rua 09 Condomínio 189 Casa 29
CEP 72.006-780 – Vicente Pires – DF – Brasil

Agradecimentos

Agradeço ao meu orientador, professor Dr. Rafael Timóteo de Sousa Júnior, que me orientou de forma profissional e amiga nas horas mais complicadas durante este trabalho e aturou tantas dúvidas e problemas relativos ao assunto e outros detalhes pertinentes à criação desta tese.

Aos Professores do Departamento de Engenharia Elétrica da UnB, Georges D.Amvame Nze, Flávio Elias, João Paulo C. Lustosa da Costa, Ugo Dias e Anderson Clayton, pelas grandes dicas, constante apoio, incentivo e amizade, essenciais para o desenvolvimento deste trabalho. Agradeço também aos membros da banca Robson de O. Albuquerque, Laerte P. de Melo, Edna D. Canedo e Rafael Rabelo Nunes.

Aos meus companheiros do Laboratório de Tecnologia da Tomada de Decisão – LATITUDE, da UnB, Daniel Alves, Alessandro Mendes, Kelly Oliveira, Andreia, Glauber, Ane e todos os outros, pelo incentivo imprescindível. Agradeço aos bolsistas dos Projetos do Laboratório LATITUDE que muito contribuíram quanto a modelos de comunicação de dispositivos, *middleware*, aplicações e demais tecnologias de IoT. Aos colegas do PPGEE, Mateus da R. Zanatta e Caio C. R. Garcez, com quem pude compartilhar as superações relativas a publicações desta tese.

Tais projetos têm apoio das Agências brasileiras de pesquisa e inovação CNPq (Projeto INCT em Segurança Cibernética 465741/2014-2), CAPES (Projeto FORTE 23038.007604/2014-69) e FAPDF (Projetos UIoT 0193.001366/2016 e SSDDC 0193.001365/2016), bem como do Ministério da Economia, anteriormente Ministério do Planejamento (TEDs 005/2016 DIPLA, 011/2016 SEST e 083/2016 ENAP) e Ministério do Trabalho (TED 001/2017 MTb), da Defensoria Pública da União (TED 066/2016 DPGU), do Gabinete de Segurança Institucional da Presidência da República (TED 002/2017 CEPESC) e do próprio Laboratório LATITUDE/UnB (Projeto SDN 23106.099441/2016-43). Durante o desenvolvimento do trabalho, fui bolsista dos projetos 005/2016 DIPLA, 001/2017 MTb e 002/2017 CEPESC; agradeço às Instituições.

Aos meus pais e irmão pelo apoio e incentivo que foi dado durante todo o tempo em que estive envolvido neste trabalho.

À minha querida esposa e filhas, por acreditar na minha capacidade e ter paciência dando total apoio e incentivo.

Meus amigos Bruno Praciano, Francisco de Caldas e Dayanne Fernandes da Cunha contribuíram de forma fundamental para a conclusão deste trabalho: meus sinceros agradecimentos.

Agradeço, acima de tudo, a Deus!

PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIoT.

Autor: Fábio Lúcio Lopes de Mendonça
Orientador: Rafael Timóteo de Sousa Júnior
Programa de Pós-graduação em Engenharia Elétrica - PPGEE
Brasília, Julho de 2019

A estrutura geral das redes de Internet das Coisas (IoT, da sigla em Inglês para a expressão *Internet of Things*) é uma questão que merece atenção. Os cenários comuns de utilização de IoT mostram um número indefinido de instâncias de IoT heterogêneas executadas de forma independente e simultânea. Em geral, cada instância de IoT opera sob um *middleware* que fornece abstrações para facilitar a integração e a comunicação de componentes heterogêneos, atuando como camada de mediação entre a camada física e a de aplicação.

Nessa situação, em função da mobilidade ou de determinações administrativas, diferentes instâncias de IoT comumente se sobrepõem no espaço físico ou na lógica de endereçamento da Internet, ocasiões em que surge a oportunidade de interoperação entre as diferentes instâncias de IoT, o que também responde a requisitos e necessidades funcionais entre tais instâncias, como a necessidade de mesclar duas ou mais instâncias, mover dados e dispositivos de uma para outra, copiar e fundir dados, substituir dispositivos, assim como de dividir uma instância existente, separando os componentes para duas ou mais instâncias.

Uma solução clássica para tais questões consiste em um determinado *middleware* centralizar o controle da interoperação, o que é caracterizado por conhecidos problemas de complexidade e inchaço do citado *middleware*, dificuldade de desenvolvimento e evolução dos componentes, dependência do componente central, etc.

Em contraposição, esta tese propõe um modelo para a interoperação de diferentes instâncias de IoT usando serviços entre entidades pares (P2P, da sigla em Inglês para a expressão *peer-to-peer*) e fornecendo operações para federar essas instâncias em cenários fixos e móveis. A arquitetura proposta suporta serviços para mesclar e dividir instâncias de IoT, bem como para mover, replicar e excluir dados da IoT, entidades de software e abstrações de dispositivos.

A proposta é concebida na forma de um modelo de interoperação com dois componentes principais, um deles devendo ser agregado na forma de biblioteca de funções a cada *middleware* controlador de instância de IoT, enquanto o outro é o serviço P2P de identificação, roteamento e busca (*lookup*) pelo qual se passam a comunicações em situação de interoperação entre duas ou mais instâncias de IoT.

A proposta é validada com o desenvolvimento e testes de um protótipo correspondente e em cenários de avaliação de suas funcionalidades e desempenho.

Palavras-chave: **Internet das coisas (IoT); *middleware* IoT; Federação IoT; Serviços *peer-to-peer* (P2P)**.

PROPOSAL OF A PEER-TO-PEER INTEROPERATION MODEL FOR INTERNET OF THINGS – P2PIoT.

Author: Fábio Lúcio Lopes de Mendonça
Supervisor: Rafael Timóteo de Sousa Júnior
Pot-graduate Program on Electrical Engineeering - PPGEE
Brasilia, Juy 2019

The general structure of the Internet of Things (IoT) networks is still a matter of great attention to research and innovation. Common IoT usage scenarios show an indefinite number of heterogeneous IoT instances executed independently and concurrently. In general, each IoT instance operates under a middleware that provides abstractions to facilitate the integration and communication of heterogeneous components, acting as a mediation layer between the physical layer and the application layer.

In this situation, depending on the mobility or administrative determinations, different instances of IoT commonly overlap in the physical space or in the Internet addressing logic, occasions when the opportunity for interoperation between the different IoT instances arises, which also responds to requirements and functional needs between such instances, such as the need to merge two or more instances, move data and devices from one to another, copy and merge data, replace devices, as well as split an existing instance by separating the components to two or more instances.

A classic solution to such questions is that a particular middleware centralizes the control of interoperation, which is characterized by known problems of complexity and swelling of the middleware, difficulty of development and evolution of the components, dependence of the central component, etc. .

In contrast, this thesis proposes a model for the interoperation of different instances of IoT using services between peer entities (P2P) and providing operations to federate these instances in fixed and mobile scenarios. The proposed architecture supports services to merge and split IoT instances, as well as to move, replicate, and delete IoT data, software entities, and device abstractions.

The proposal is designed in the form of an interoperation model with two main components, one of which must be aggregated in the form of a function library to each IoT instance controller middleware, while the other is the P2P identification, routing and lookup service through which communications are made in a situation of interoperation between two or more instances of IoT.

The proposal is validated with the development and testing of a corresponding prototype and in scenarios evaluating its functionalities and performance.

Keywords: **Internet of Things (IoT); IoT middleware; IoT Federation; Peer-to-peer Services (P2P)**.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TERMOS E SIGLAS

API	*Application Programming Interface*
DDS	*Distributed data service*
GPS	*Global Positioning System*
IaaS	*Infrastructure as a Service*
IDE	*Integrated Development Environment*
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	*Internet Protocol Version 4*
IPv6	*Internet Protocol Version 6*
JSON	*JavaScript Object Notation*
LPS	*Local Positioning System*
P2P	*Peer-to-Peer* (Ponto a Ponto)
REST	*Representational State Transfer*
RFID	*Radio Frequency identification*
RNP	Rede Nacional de Ensino e Pesquisa
SOA	*Service Oriented Architecture*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
UUID	*Universal Unique Identifier*
VM	*Virtual Machine*
WSN	*Wireless Sensor Networks*

Capítulo 1

Introdução

As tecnologias que compõem a Internet das Coisas (IoT) vêm permitindo a construção de aplicações de grande impacto, que estabelecem inter-relações e fluxos de informação com os mais diversos objetos físicos, em cenários fixos e móveis. A quantidade de dispositivos IoT está em contínuo crescimento, avançando para a ordem de bilhões de dispositivos conectados em um futuro próximo. Tal número, associado à variedade de serviços providos pelos dispositivos e aplicações de IoT existentes, constitui um cenário desafiador para o gerenciamento desses dispositivos e serviços caso não sejam utilizadas abordagens avançadas para o seu registro e organização em redes IoT, havendo na literatura uma variedade de propostas de arquiteturas e modelos para responder a tal questão [6].

Como o conceito de IoT envolve plataformas de hardware e sistemas operacionais potencialmente diferentes, assim como *middleware*, protocolos de comunicação, modelos de dados, tipos de aplicativos e políticas de segurança, entre outros componentes, não nos referimos a uma Internet das Coisas única, homogênea e coesa. Em vez disso, uma definição mais realista refere-se ao conceito de várias instâncias de redes IoT em execução paralela e simultânea. Tais instâncias independentes se configuram também em razão de separação espacial ou de domínio administrativo, mesmo quando se considera a utilização de um conjunto de hardware e software homogêneo.

Considerando tal situação, existe a possibilidade, e mesmo a necessidade, de interações de uma instância de IoT para outra, seja entre componentes específicos (por exemplo, um dispositivo operando simultaneamente em mais de uma rede IoT), seja na forma de operações envolvendo a integralidade dos elementos de duas ou mais dessas instâncias de IoT (por exemplo, no caso de uma instância de rede de IoT móvel se aproximar de outra instância fixa ou móvel, levando a uma possível necessidade de mesclar as instâncias ou, vice-versa, para dividir uma instância existente).

Nesta tese, propomos um modelo de interoperação projetado para servir, de fato, às várias encarnações ou instâncias de IoT simultâneas e paralelas. O modelo aqui proposto, P2PIoT, visa prover meios e serviços para que as interações ocorram de uma forma distribuída, entre pares de instâncias de IoT, ou seja, de forma *peer-to-peer* – P2P.

1.1 Motivação

A organização típica para cada instância de IoT inclui dispositivos IoT, tanto aqueles com restrições de recursos (sensores simples e atuadores) quanto os relativamente sem restrições (objetos inteligentes). Inclui também *gateways* IoT que fornecem conversões de protocolo e serviços de mediação para alguns desses dispositivos, *middleware* IoT e aplicações IoT, como mostrado na Figura 1.1a. Nessa organização, o *middleware* IoT gerencia efetivamente os vários componentes heterogêneos que devem interconectar e interoperar. Especificamente, o *middleware* IoT é uma camada de software entre a camada física de dispositivos e suas interfaces e a camada de aplicativo que integra a lógica e o controle da instância da IoT. Haja vista que há requisito de fácil integração e comunicação entre os componentes heterogêneos, o *middleware* IoT é necessário para fornecer um conjunto de abstrações de programação, incluindo descoberta de dispositivos, acesso a serviços e tarefas de administração.

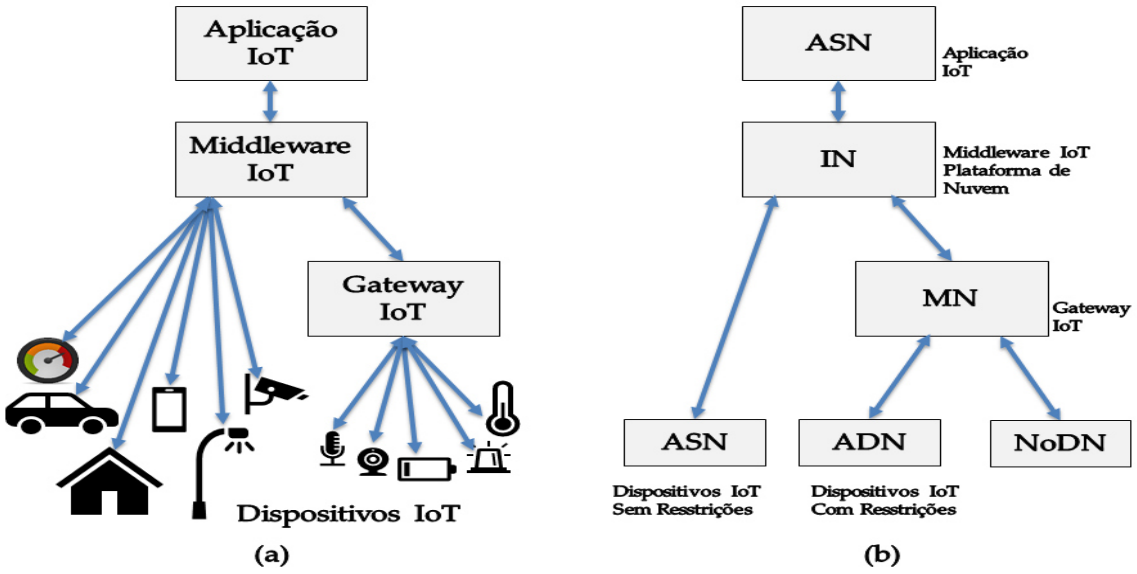


Figura 1.1: Organização comum de uma Instância de IoT (**a**); e sua correspondente configuração funcional oneM2M (**b**). Adaptado de [1]

A Figura 1.1b também apresenta o mapeamento de uma instância da IoT comum para a arquitetura funcional oneM2M [7], valendo notar que o Consórcio oneM2M é um dos principais organismos promotores de padrões para comunicações máquina a máquina (M2M) e Internet das Coisas. De acordo com essa figura, no modelo oneM2M, as instâncias de IoT compreendem nós de infraestrutura (IN), nós intermediários (MN), nós de serviço de aplicativo (ASN), nós dedicados de aplicativos (ADN) e nós não-oneM2M (NoDN). Os nós designados como NoDNs correspondem a sensores e atuadores simples, enquanto os ADNs são objetos inteligentes. Os ASNs podem ser objetos inteligentes ou dispositivos de aplicativos clientes ou *gateways* para dispositivos simples. MNs são dispositivos irrestritos. INs são plataformas ou servidores em nuvem para os principais módulos de *middleware*.

Considerando tal organização da instância individual de IoT, os cenários comuns de utilização mostram um número indefinido de instâncias de IoT heterogêneas executadas de forma independente e simultânea. Em geral, cada instância da IoT opera sob um *middleware* que fornece um conjunto de abstrações para facilitar a integração e a comunicação de componentes heterogêneos, atuando como uma camada de mediação entre a camada física e a camada de aplicação.

Um artigo de levantamento recente [8] corrobora muitos trabalhos anteriores que fizeram parte do estudo de trabalhos relacionados desta tese, trabalhos estes que afirmam que a IoT tem as seguintes características: 1) descentralização de sistemas/aplicações de IoT, 2) diversidade de dispositivos e sistemas de IoT, 3) heterogeneidade de dados de IoT e 4) complexidade de rede. Essa mesma referência bibliográfica indica que essas características juntas acarretam os seguintes desafios de pesquisa: heterogeneidade dos sistemas de IoT, baixa interoperabilidade entre esses sistemas, restrições de recursos dos dispositivos de IoT, bem como vulnerabilidades de privacidade e segurança.

Considerando que a interoperabilidade é definida como a capacidade de interagir com sistemas físicos e trocar informações entre sistemas IoT, ou seja, com uma conceituação similar à que é adotada no âmbito desta tese, o mesmo artigo [8] argumenta, quanto à Interoperabilidade entre dispositivos IoT, sistemas de IoT e setores das aplicações de IoT, que a interoperabilidade de IoT pode ser alcançada por meio de uma camada de software *blockchain* em cima de uma rede de sobreposição P2P com acesso uniforme em diferentes sistemas de IoT.

Embora tenha base conceitual semelhante à desta tese, enquanto o citado artigo permanece no raciocínio conceitual, esta tese se desenvolve além da conceituação básica, com a especificação de serviços de interoperabilidade e sua realização sobre um sistema de busca P2P.

Nesse sentido, o desenho do modelo aqui proposto, com módulos articulados para serviços de interoperabilidade e a interface de programação para serviços de busca P2P, contempla assim as duas contribuições fundamentais desta tese. Outras contribuições complementares vêm do processo de validação, que inclui o desenvolvimento de um protótipo para avaliar os serviços de interoperação, bem como a discussão da escalabilidade e desempenho na presença de um número variável de dispositivos, tanto restritos quanto irrestritos no que se refere às capacidades computacionais e de comunicação.

Vale, entretanto, notar que, entre os desafios citados por [8], esta tese não aborda temas relativos aos aspectos de confidencialidade, incluindo confidencialidade da informação pessoal ou privacidade, e integridade, mas considera que a disponibilidade de serviços e de dados é aspecto diretamente vinculado às proposições de funcionalidades de interoperação para IoT. Embora evidencie-se que uma abordagem geral da segurança de IoT, integradora dos aspectos de confidencialidade, integridade e disponibilidade, seja um requisito imprescindível na participação de dispositivos, *middleware* e aplicações em uma ou várias instâncias de IoT, seguimos nesta tese o argumento de [9] segundo o qual a lógica de funcionamento da aplicação deve ser separada das políticas de execução da segurança.

Em consequência, esta tese é dedicada à investigação de uma proposta de funcionalidade

para a interoperação de IoT, de modo a permitir o teste da efetividade e do desempenho dessa funcionalidade, considerando para outros estudos o aspecto geral da segurança, mas pressupondo que medidas de segurança distribuídas de IoT [10] estejam ou possam ser estabelecidas em cada instância de IoT, tais como a proteção das comunicações [11], a autenticação contínua [1], a prevenção e detecção de intrusões [12]. Ademais, o modelo de interoperação proposto nesta tese, sendo da classe P2P, adequa-se ao acoplamento de extensões no sentido de integrar o consenso e o registro das transações de modo distribuído, de modo análogo ao protocolo *blockchain* e outros algoritmos de consenso.

Por outro lado, de acordo com [13], um grande interesse nas comunidades de pesquisa de IoT consiste em promover interconectividade entre plataformas de IoT com base em federação, unificação e interoperabilidade semântica entre vários domínios de IoT. Contudo, tal artigo destaca que as soluções que dependem exclusivamente da adoção de infraestruturas e serviços em nuvem provavelmente não conseguirão acompanhar os problemas esperados de escalabilidade e congestionamento de rede em futuros cenários de IoT, problemas estes que na verdade têm soluções promissoras quando se considera os potenciais da computação na borda e nos dispositivos (*egde*, *fog*).

Por essa razão, reiteramos que, na presente tese, a proposta é prover um serviço de busca P2P acoplado com serviços de interoperação entre instâncias de IoT, de forma independente da escolha de um ambiente de nuvem ou de computação na borda, ou mesmo misto, para implantação dos módulos de cada instância IoT. Conforme já apresentado, a essas duas contribuições agregam-se também as contribuições do processo de validação, observando que o desenvolvimento de um protótipo para avaliar os serviços de interoperação, permite a discussão da escalabilidade e do desempenho da solução P2P, notando que esta solução naturalmente se adequa à computação na borda e nos dispositivos mostrando-se, quanto a tais aspectos, mais promissora que as soluções centralizadas e organizadas somente com processamento em nuvem.

Assim, em síntese esta tese propõe um modelo para a interoperação de diferentes instâncias de IoT usando um serviço de busca P2P ao qual se sobrepõem serviços de interoperação de IoT. Dessa forma, os serviços de busca e interoperação são integrados ao *middleware* IoT para fornecer operações destinadas a federar as instâncias heterogêneas de IoT em cenários fixos e móveis. O modelo proposto suporta serviços para mesclar e dividir instâncias de IoT, bem como para mover, replicar e excluir dados da IoT, ou de suas entidades de software e abstrações de dispositivos. A proposta é validada testando um protótipo desenvolvido e discutindo suas funcionalidades e desempenho.

1.2 Objetivo

O principal objetivo deste trabalho é propor um modelo de interoperação em arquitetura peer-to-peer (P2P) entre middlewares de IoT, de modo a permitir funcionalidades de interoperação tais como agrupar, separar, copiar, apagar (*Merge*, *Split*, *Copy*, *Delete*) relativas a instâncias de IoT,

seus dispositivos e dados.

O embasamento do trabalho prescinde de estudos sobre arquitetura e *middlewares* de IoT, modelos de comunicação P2P, auto-registro de dispositivos e outros temas pertinentes que são tratados no estudo de trabalhos relacionados.

1.3 Objetivos Específicos

Para alcançar o objetivo geral desta proposta, os seguintes objetivos específicos foram definidos:

- Estudar e especificar funcionalidades de interoperação para instâncias heterogêneas de IoT.
- Propor a arquitetura P2P para suporte à interoperação em IoT.
- Apresentar um protótipo envolvendo a tecnologia P2P e funcionalidades de IoT, oferecendo uma realização da arquitetura proposta para permitir a avaliação dos resultados.
- Realizar validação da proposta, por intermédio de baterias de testes das funcionalidade de interoperação, seu desempenho e comparação com uma arquitetura centralizada.

1.4 Contribuição do Trabalho

O modelo aqui proposto articula-se em dois módulos, sendo o primeiro de serviços de interoperabilidade específicos de IoT e o segundo, de interface de programação para serviços de busca P2P, o que define as duas contribuições fundamentais buscadas nesta tese.

Além disso, outras contribuições são buscadas através da avaliação do modelo proposto, o que inclui o desenvolvimento de um protótipo para testar e validar os serviços de interoperação, bem como a discussão da escalabilidade e desempenho na presença de um número variável de dispositivos, tanto restritos quanto irrestritos no que se refere a capacidades computacionais e de comunicação.

1.5 Publicações Vinculadas ao Trabalho

- [14] Zanatta, M. R.; Mendonca, F. L. L.; Antreich, F.; Lima, D. V.; Miranda, R. K.; Del Galdo, G.; da Costa, J. P. C. L. Tensor-based time-delay estimation for second and third generation global positioning system. Digital Signal Processing, v. 1, p. 1, 2019.
- [15] Torres, J. A. S.; Silva, D. A.; Mendonca, F. L. L.; Barbosa, N. F.; De Sousa Jr, Rafael T. Melhoria da precisão dos indicadores na governança digital de serviços públicos à vista da

análise de bases de dados de empregabilidade. Inclusão Social (Online), v. 12, p. 166-182, 2019.

- [16] Silva, D. A.; Machado, P. L.; Coelho, V. C. G.; Barbosa, R. V.; Mendonca, F. L. L.; Santos, D. P.; De Sousa Jr, Rafael T. Produção de indicadores de empregabilidade com base em técnicas de mineração de Big Data e Business Intelligence. Inclusão Social (Online), v. 12, p. 141-155, 2019.

- [17] Justino Garcia Praciano, Bruno; Carvalho Lustosa Da Costa, Joao Paulo; Abreu Maranhao, Joao Paulo; Lopes De Mendonca, Fabio Lucio; De Sousa Junior, Rafael Timoteo; Barbosa Prettz, Juliano. Spatio-Temporal Trend Analysis of the Brazilian Elections Based on Twitter Data. In: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 2018, Singapore. 2018 IEEE International Conference on Data Mining Workshops (ICDMW). Piscataway: IEEE Xplore, 2018. v. 1. p. 1355.

- [18] Martins, Lucas; Filho, Francisco L. De Caldas; Araújo, Ingrid Palma; de Mendonça, Fábio L. L.; da Costa, João Paulo C. L.; de Sousa Júnior, Rafael T. Design and Evaluation of a Semantic Gateway Prototype for IoT Networks. In: Companion the10th International Conference, 2017, Austin. Companion Proceedings of the10th International Conference on Utility and Cloud Computing - UCC '17 Companion. New York: ACM Press, 2017. p. 195.

- [19] Filho, Francisco L. De Caldas; Martins, Lucas M. C. E; Araújo, Ingrid Palma; Mendonca, F. L. L.; Costa, J. P. C. L.; De Sousa, Jr., Rafael T. Gerenciamento de serviços IoT com gateway semântico. In: Conferências Ibero-Americanas WWW/Internet e Computação Aplicada 2017, 2017, Vilamoura. Lisboa: IADIS Press, 2017. v. 1. p. 199-206.

- [2] De Melo Silva, Caio César; Caldas, F. L.; Machado, F. D.; Mendonca, F. L. L.; De Sousa, Jr., Rafael T. Proposta de auto-registro de serviços pelos dispositivos em ambientes de IoT. In: XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2016), 2016, Santarém. Rio de Janeiro: SBrT, 2016. v. 1. p. PS-7.

- (Artigo aceito) Fábio L. L. de Mendonça, Dayanne F. da Cunha, Bruno J. G. Praciano, Mateus da Rosa Zanatta, João Paulo C. L. da Costa, Rafael T. de Sousa Jr. P2PIoT: A Peer-To-Peer Communication Modelfor the Internet of Things. Disponível em: https://wcnps.redes.unb.br/submission.php

- (Artigo aceito) Fábio L. L. de Mendonça, Daniel Valle de Lima, Mateus da Rosa Zanatta, João Paulo C. L. da Costa, Ricardo Kehrle Miranda, Andre L. F. de Almeida e Rafael T. de Sousa Jr. Estimação de Componentes de Multipercurso para Receptor Tensorial de GPS de 2a e 3a Geração. XXXVII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, 2019, Petrópolis - RJ. Disponível em: http://sbrt.org.br/sbrt2019/wp-content/uploads/2019/05/SBrT2019CFPPor-ext.pdf.

- (Artigo aceito) Daniel Alves da Silva, José Alberto Sousa Torres, Alexandre Pinheiro, Francisco L. de Caldas Filho, Fabio L. L. Mendonça, Bruno J. G Praciano, Guilherme Oliveira Kfouri, Rafael T. de Sousa Jr. Inference of driver behavior using correlated IoT data from the vehicle telemetry and the driver mobile phone. In: Preproceedings of the Federated Conference on Computer Science and Information Systems, pages 497 – 501. Disponível em: https://annals-csis.org/proceedings/2019/pliks/263.pdf.

- (Artigo aceito) Caio C. R. Garcez, Daniel Valle de Lima, Ricardo Kehrle Miranda1, Fábio L L de Mendonça João Paulo C. L. da Costa e Rafael Timóteo de Sousa Jr. Técnicas Tensoriais Baseadas em Rastreamento de Subespaços Aplicadas a Sistemas GNS. XXXVII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, 2019, Petrópolis - RJ. Disponível em: http://sbrt.org.br/sbrt2019/wp-content/uploads/2019/05/SBrT2019CFPPor-ext.pdf.

1.6 Metodologia

No sentido de alcançar os objetivos desta tese, o projeto de pesquisa constitui-se das fases iniciais de identificação e caracterização do problema da interoperabilidade em IoT, levantamento de uma hipótese de que uma solução distribuída, com entidades de igual poder atuando como pares, abarcaria os serviços necessários, com a escalabilidade e o desempenho requisitados.

Às fases iniciais, seguem-se fases de concepção, tanto da estrutura geral quanto dos módulos do modelo P2P para interoperação em IoT, levando às fases finais com o desenvolvimento de um protótipo de hardware e software, destinado a permitir a avaliação da proposta em cenários de teste das funcionalidades de interoperação P2P para IoT e averiguação de sua escalabilidade e desempenho.

Isso posto, em linhas gerais o desenvolvimento desta tese corresponde à metodologia de pesquisa construtiva ou, da denominação em Inglês, *Design Science* [20], uma metodologia considerada como particularmente bem adaptada a sistemas de informação, com relação aos quais as metodologias mais clássicas têm alcance limitado.

Segundo a metodologia de pesquisa *Design Science*, as atividades de pesquisa consistem em seis fases principais, sendo elas:

1. Construir estrutura conceitual.
2. Estabelecer a formalização da estrutura.
3. Desenvolver arquitetura do sistema.
4. Analisar e projetar o sistema.
5. Construir o protótipo do sistema.

6. Observar e avaliar o sistema.

Assim, há correspondência bastante direta entre tais fases e aquelas realizadas, o que também determina a organização da tese, conforme apresentado a seguir.

1.7 Organização do Trabalho

Esta tese é composta por cinco capítulos, incluindo este de introdução.

O capítulo 2 trata de trabalhos relacionados e bases conceituais da tese, o que inclui uma breve revisão de arquiteturas de IoT (mecanismos de controle e fluxos de dados em IoT, esquemas de auto registro de dispositivos, estruturas de *middleware* e *gateways* de IoT) e protocolos para registro, roteamento e busca de itens em redes sobrepostas P2P.

No capítulo 3, o modelo de interoperação para instâncias de IoT é apresentado, constituído a contribuição central da tese. São apresentados os requisitos de funcionalidades de interoperação em IoT, o modelo proposto e seus dois componentes, assim com são descritos os processos operativos que compõem os serviços de interopeção propostos.

O capítulo 4 destina-se à avaliação da proposta, descrevendo o protótipo desenvolvido, os cenários de validação das funcionalidades de interoperação em IoT, bem como a escalabilidade, desempenho e comparação com um modelo centralizado, servindo assim à discussão de resultados obtidos.

O capítulo 5 apresenta a conclusão deste trabalho e propõe trabalhos futuros.

Capítulo 2

Conceitos e Fundamentação Teórica

Este capítulo procura abordar os conceitos mais pertinentes à construção do Modelo P2PIoT proposto nesta tese. Discute também temas vinculados ao desenvolvimento de um protótipo correspondente ao modelo proposto, protótipo este destinado à utilização no processo de validação do P2PIoT. Este capítulo coloca-se assim em conformidade com a metodologia adotada, ou seja, de apresentar os elementos que permitam a compreensão da concepção modular e funcional do P2PIoT (tratada no Capítulo 3) seguida de sua validação experimental com a devida discussão dos resultados (assunto do Capítulo 4).

Sendo um modelo voltado à interoperação de diferentes instâncias de IoT, é importante tratar inicialmente de características gerais do paradigma de Internet das Coisas, mas já adiantando precisões, cujo entendimento vem se estabilizando na literatura, acerca da estrutura geral dos modelos e de fluxo de comunicação entre dispositivos e aplicações de IoT, com intermediação do *middleware* de IoT.

Considerando que a quantidade e a variabilidade dos dispositivos é um importante fator em IoT, e que é virtualmente impossível às aplicações terem registro prévio de todos os possíveis dispositivos que possam participar da IoT, e considerando que essa questão se torna mais complexa à medida em que se pensa na interoperação entre diferentes instâncias de IoT, é importante tratar da possibilidade de auto registro de dispositivos. No processo de auto registro, cabe ao dispositivo informar suas capacidades (dados e serviços) de modo que o *middleware* de IoT tenha a possibilidade de criar uma visão abstrata dos dispositivos. Tal visão abstrata, ou seja, na forma de estrutura de dados acoplada com serviços de software, pode ser geralmente usada pelas aplicações em uma determinada instância de IoT ou pode ser ativada nas funcionalidades de interoperação de uma instância para outra, de forma independente da constituição física dos dispositivos, seus modos de armazenamento, seus protocolos de monitoração, acionamento e comunicações.

Por outro lado, é necessário tratar dos trabalhos relacionados sobre o tema da interoperabilidade, particularmente quanto as funcionalidades requisitadas e possíveis arcabouços para a execução das respectivas operações.

Considerando que, por hipótese adotada nesta tese, um arcabouço de interoperação entre enti-

dades pares, o denominado modelo P2P, promete uma solução efetiva para a questão, este capítulo trata também de sistemas de busca P2P, especialmente apresentado de maneira sucinta o sistema *Chord*[4], considerado em geral um paradigma nesse domínio do conhecimento. Outro interesse desse sistema é que existe na forma de código-fonte disponível para a pesquisa, o que permite sua utilização no desenvolvimento do protótipo, que nesta tese é destinado à utilização no processo de validação da proposta P2PIoT.

Um problema fundamental que confronta os aplicativos peer-to-peer é localizar eficientemente o nó que armazena um determinado item de dados. Este artigo apresenta o Chord, um protocolo de pesquisa distribuído que trata desse problema. Chord fornece suporte para apenas uma operação: dada uma chave, ela mapeia a chave em um nó. Os dados podem ser facilmente lidos em cima do Chord, associando uma chave a cada item de dados e armazenando o par chave / item de dados no nó para o qual a chave mapeia. O acorde se adapta de maneira eficiente à medida que os nós entram e saem do sistema e podem responder mesmo se o sistema estiver mudando continuamente. Resultados de análises teóricas, simulações e experimentos mostram que o Chord é escalável, com custo de comunicação e o estado mantido por cada nó escalando logaritmicamente com o número de nós Chord.[4]

Organizam-se assim, de forma correspondente à descrição acima, as seções a seguir, tratando de temas fundamentais ao problema objeto do trabalho e à hipótese adotada.

2.1 Paradigma IoT e Estrutura Geral do *Middleware* de IoT

O termo "Internet das Coisas" foi mencionado pela primeira vez por Kevin Ashton em uma apresentação de 1999 [21], mas desde então não tem uma definição universalmente aceita e muitas vezes, as definições de temas associados são conflituosas. Por essa razão, ao invés de trazer uma definição estrita, neste capítulo são discutidas várias conceituações do tema, mesmo que conflituosas, mas que resultam em uma síntese de visões a partir de pontos de vistas diferenciados.

Em várias referências da literatura, encontra-se um significado excessivamente generalizador de IoT como "uma rede mundial de objetos interconectados que são endereçados de forma única, baseada em protocolos de comunicação padrão" [22]. Argumentaremos no texto a seguir que não se pode falar a rigor em uma rede mundial para IoT, nem mesmo na possibilidade de um mesmo conjunto de protocolos em todos os casos de IoT. Entretanto, vale como ideia geral introdutória do termo objeto, também denominado de dispositivo IoT, que corresponde à coisa na expressão Internet das Coisas.

A coisa na IoT, ou objeto, é uma entidade orgânica ou inorgânica, integrada de hardware e software, que contém e coordena dispositivos inteligentes. Por exemplo, corresponde a um automóvel equipado com sensores que avisa ao motorista que alguns componentes não estão funcionando, um animal com um chip de sensoriamento ou qualquer objeto que possua um método de identificação, como um endereço IP, e que possa transmitir dados fazendo uso de uma rede de comunicação de dados [23] do tipo internet.

Em decorrência dessa conceituação, não havendo nenhuma outra restrição conceitual, observa-se então que a IoT levanta desde o início a questão de um elevado número de objetos heterogêneos envolvidos nos processos de identificação, representação e trocas de informação, o que constitui um tema desafiador.

Em particular, vale notar os objetos vinculados a tecnologias de redes de sensores sem fio (*Wireless Sensor Networks* – WSN), hoje difundidos em várias áreas da vida moderna e oferecendo a capacidade de medir, inferir e entender indicadores ambientais, desde aqueles da ecologias dos recursos naturais até os ambientes urbanos artificiais. Na IoT, tais sensores e atuadores se integram ao ambiente ao nosso redor e geram informações que podem ser compartilhadas pelas mais diversas plataformas computacionais e de redes para uso das mais diversas aplicações.

Resultante da adaptação de uma variedade de tecnologias sem fio, como etiquetas RFID, sensores e atuadores embutidos, essa forma de rede contribui para que a IoT seja uma das facetas principais da computação ubíqua na *web*, em que a necessidade de dados sob demanda usando consultas intuitivas sofisticadas aumenta significativamente [24].

Essa conceituação de IoT na visão de objeto contempla assim uma variedade de objetos com presença pervasiva, incluindo itens que podem ser identificado por radiofrequência (*Radio Frequency Identification* - RFID), mas também sensores, celulares e outros aparelhos [25]. Tal conceito se estender ainda para coisas identificadas ou conectadas por vários protocolos de comunicação de dados, como *WiFi* e *bluetooth*.

Entretanto, considerada a heterogeneidade dos dispositivos e a variabilidade da informação a eles associada, um importante aspecto adicional é a visão da IoT orientada à semântica [26], o que envolve os temas relacionados à representação, ao armazenamento, à interconexão, à busca e à organização de informação gerada pela IoT. Neste contexto, os conceitos e tecnologias da semântica (*linked data*, ontologias, entre outros) possuem papel importante para modelar, descrever e analisar os dados gerados pela IoT [27].

Integrando então objetos que podem manipular o ambiente a partir do significado da informação, vê-se a IoT também como uma expansão dos serviços atuais da Internet, o que representa um grande desafio científico e tecnológico, no sentido de acomodar todo e qualquer objeto do mundo, tanto aqueles legados ou os que serão desenvolvidos num futuro próximo.

Nessa linha de raciocínio, observa-se que, do ponto de vista técnico, a IoT vem resultando de vários desenvolvimentos complementares que associam o mundo computacional, virtual e o físico, o que inclui inovações em domínios variados tais como as que são citadas por [28] [29], aqui resumidas:

- Identificação: os objetos são identificados de modo unívoco;
- Endereçamento: os objetos podem localizar-se e endereçar-se através de serviços de descoberta e consulta, sendo consultados e configurados remotamente;
- Comunicação e cooperação: os objetos têm a capacidade de interconexão via Internet, o

que permite usar a rede para obter dados, serviços e atualizar o estado dos objetos;

- Sensoriamento: os objetos coletam informações sobre o ambiente com sensores, registrando, encaminhando a informação ou reagindo diretamente a eventos do ambiente;
- Atuação: os objetos contêm atuadores para manipular o ambiente;
- Processamento de informação embutido: os objetos, ditos inteligentes, possuem um processador ou microcontrolador, além da capacidade de armazenamento; tais recursos podem ser utilizados para processar e interpretar as informações dos sensores, ou para dar aos objetos uma memória de como eles foram utilizados;
- Localização: os objetos inteligentes estão cientes da sua localização física ou podem ser localizados;
- Interfaces para usuário: os objetos inteligentes podem se comunicar com os indivíduos de modo apropriado, de forma direta ou indireta, através de *middleware* ou aplicações de IoT.

2.1.1 Caracterização dos Dados de IoT

Da perspectiva de dados, observa-se já no presente momento que o volume, a variabilidade e a velocidade de surgimento dos dados gerados, armazenados e processados constituem um desafio importante, talvez sem precedentes na história da Internet. Há uma percepção generalizada na literatura de que tal situação deve perdurar.

Grande parte dos dados geridos em IoT representam fenômenos do mundo físico, haja vista a origem em dispositivos sensores e atuadores. De acordo com [30], e discussões adicionais desta tese, os dados de IoT podem ser classificados como segue:

- Dados de Identificação RFID: utilizados para a identificação e o rastreamento de objetos através ondas de rádio, na forma de rótulos eletrônicos que podem ser inseridos nos objetos e utilizados para transmitir e receber informação.
- Dados de endereçamento e identificação única: em geral, os objetos IoT precisam ser identificados com endereços IP para serem alcançáveis na Internet, mas também identificação própria na abstração da IoT, como, por exemplo, com Identificação Única Universal (*Universal Unique Identifier* – UUID). O número de identificadores necessários crescerá portanto na mesma proporção da quantidade dos objetos IoT. Quando se coloca a questão da interoperação entre diferentes redes IoT, a identificação deve ainda considerar a identificação cruzada dos objetos e possíveis conflitos de identificadores e endereços IP, assuntos resolvidos nesta tese graças aos elementos de identificação próprios do sistema P2P que é um dos módulos da proposição.

- Cópias da base informacional de um dispositivo para outro. Entre as funcionalidades de interoperação propostas nesta tese, é prevista uma operação de cópia em uma federação P2P de IoT;

- Metadados dos objetos, processos e sistemas: os metadados registrados pelos objetos, processos e sistemas participantes em um ambiente IoT são essenciais para possibilitar aos usuários que encontrem e acessem os dados apropriados. O modelo proposto nesta tese inclui uma abstração dos objetos que provê o acesso aos metadados na forma de serviços vinculados aos objetos.

- Dados de posicionamento: a localização de um objeto dentro de um ambiente IoT pode ser feita através do Sistema de Posicionamento Global (*Global Positioning System* – GPS) ou Sistema de Posicionamento Local (*Local Positioning System* – LPS). Os dados de posicionamento são importantes para a IoT, uma vez que pode descrever se um objeto ou grupo de objetos é estático ou móvel, o que terá implicações na automatização das funcionalidades de interoperação propostas nesta tese, a partir dos cálculos de posicionamento e proximidade das redes IoT envolvidas. Em todo caso, a utilização de dados de posicionamento é importante aspecto do desempenho da proposta, pois representa um desafio conjunto de atraso, precisão e complexidade algorítmica para os *middleware* de IoT que estiverem em interoperação;

- Dados de sensores: uma das fontes de dados para IoT são as redes de sensores que tornam possível a captura de grandes quantidades de dados de forma rápida e em tempo real.

- Dados históricos: conforme já comentado, trata-se de desafio sem precedentes o volume, variabilidade e velocidade de dados que são capturados pelos dispositivos/sensores de IoT. Tais dados precisam ser armazenados, com a passagem do tempo, se tornam históricos, amplificando o desafio, haja vista os requisitos de preservação, armazenamento, inferência, auditabilidade.

Constata-se assim, o desenrolar no tempo das atividades das gestão de dados em IoT e as consequentes questões a serem resolvidas no tratamento e armazenamento desses dados em tempo real e em tempo diferido.

Vale comentar que se tratam aqui de desafios relativos a séries temporais de dados decorrentes das características temporais da IoT, uma vez que o indicador do tempo está presente de forma explícita ou implícita nos dados gerados em IoT [31] [32] [33].

Uma série temporal é definida como uma sequência de dados numéricos de uma mesma variável, onde cada item representa um valor em um determinado ponto do tempo [34]. Assim, ainda que em um ambiente IoT os dados sejam coletados em intervalos de tempo variável, acumulam-se dados regulares ao longo de um determinado período, o que permite analisar as caraterísticas estatísticas das séries, para efeito de inferências, compactações, preservação de itens de dados individuais, etc.

Segundo [35], há dois tipos de séries temporais, a saber: estacionárias e não estacionárias. As primeiras flutuam em torno de uma mesma média ao longo do tempo e as segundas têm média que variam ao longo do tempo. Ambas, estão presentes em IoT haja vista que as medições feitas pelos objetos podem ou não variar em torno a uma média, o que dependerá do tipo de objeto que faz as medições, das grandezas medidas e das fontes dos dados físicos.

Em um trabalho precedente e próximo à presente tese [36], tais assuntos são tratados de maneira sistemática e são propostas soluções na forma de um serviço distribuído de dados (*distributed data service* – DDS) provido na forma de um *middleware* para IoT. Tal serviço pode ser opcionalmente utilizado em conjunto com a proposta de Modelo P2PIoT desta tese.

2.1.2 Conceito de Instância de IoT

Argumentamos anteriormente que não faz sentido nos referimos a apenas uma única, homogênea e coesa Internet das Coisas, ou seja, uma rede mundial de objetos interconectados que são endereçados de forma única, baseada em protocolos de comunicação padrão da Internet.

A heterogeneidade dos hardware e software dos dispositivos, canais de comunicação, protocolos de acesso, sistemas operacionais, *middleware*, bases de dados, representações abstratas da informação, estrutura e funcionamento das aplicações, bem como nos modos de administração, resultaram em uma situação em que constata-se a existência de várias redes de IoT operando simultaneamente e em paralelo, de modo que, em consequência nos referimos nesta tese ao conceito de várias encarnações de Internet das Coisas que denominamos instâncias de IoT, retomando o conceito de um trabalho precedente próximo da presente tese [2].

Para ilustrar a simultaneidade e paralelismo entre instâncias de IoT, nos referimos à Figura 2.1, que ilustra uma situação comum de um dispositivo celular multifuncional transportado por uma pessoa que dirige um veículo entre sua casa e o trabalho. Tal dispositivo participa simultaneamente de diversas instâncias de IoT, sendo cada instância na verdade uma rede sobreposta (*overlay network*) sobre a infraestrutura fixa e móvel da Internet.

Frisamos aqui o conceito de sobreposição, pela razão de a instância de IoT manter, normalmente sob a égide do *midlleware*, uma estrutura interna de identificação de dispositivos e aplicações, bem como um controle de fluxo de informações exclusivo para tais dispositivos, o que constitui para efeito de operações de IoT um grupo delimitado e fechado, embora utilize os recursos da Internet para comunicações. Essa sobreposição se faz independente do fato de algum dos componentes participar de algum outro tipo de aplicação ou sistema distribuído.

Ainda com referência à Figura 2.1, o dispositivo móvel tem diferentes papéis em cada instância de IoT e a situação se modifica continuamente em função de diversos fatores, tais como o movimento, a capacidade energética restante, as decisões dos usuários, etc. O dispositivo pode estar presente simultaneamente em diversas instâncias de IoT. A existência e convivência de tais instâncias envolvem potencialmente diferentes plataformas de hardware e sistema operacional, *middleware* de IoT, protocolos de comunicação, modelos de dados, políticas de segurança, tipos

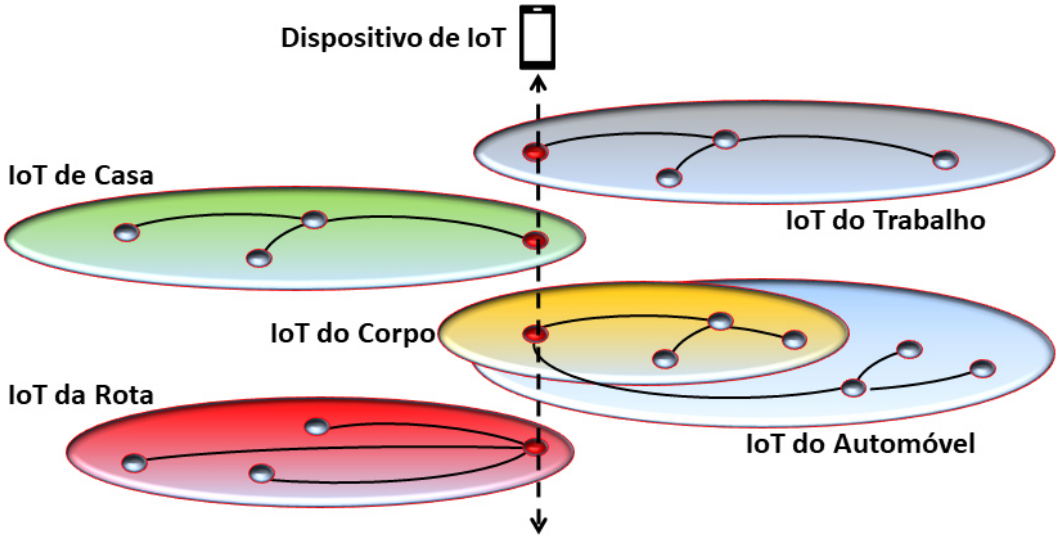


Figura 2.1: Diferentes instâncias em que participa um dispositivo de IoT [2]

de aplicações, haja vista que cada instância tem sua configuração particular no que se refere a cada um desses elementos.

O trato da heterogeneidade e da escalabilidade são fundamentais em um sistema complexo e dinâmico como esse, levando à necessidade de pouca intervenção humana, devendo os dispositivos oferecer auto-capacidades, notadamente elevado grau de autonomia de configuração, auto-organização e auto-adaptação a vários cenários, auto-reação a eventos e estímulos, e auto-processamento de enormes quantidades de dados trocados [37].

Nessa situação, podendo simultaneamente ofertar diversos serviços, sob condições diferenciadas, e que podem se modificar a qualquer instante, é importante haver sensibilidade ao contexto, tanto pelo dispositivo quanto pelas possíveis aplicações em que o dispositivo esteja contribuindo.

A heterogeneidade das várias instâncias implica em potencial sobrecarga de artefatos de software, módulos de comunicação, aplicações e tarefas administrativas no dispositivo, com implicações na utilização dos seus recursos de energia, armazenamento, memória, capacidade de processamento e de comunicações e consequências quanto ao desempenho.

Tal problemática não é tratada nesta tese, mas apenas consideramos que o dispositivo é visto em cada instância de IoT pelo respectivo *middleware* para interagir no interior da instância de IoT com outros dispositivos e aplicações dessa mesma instância (interoperabilidade interna à instân-

cia). Isso permite que, a cada *middleware*, sejam agregados módulos que permitam a interação com os outros *middleware* das demais instâncias de IoT provendo funcionalidades de interoperação inter-instâncias de IoT.

Por tal razão, é interessante tratar mais detidamente do conceito de *middleware*de IoT, o que é objeto das próximas seções.

2.2 Precisões Acerca do Conceito de *Middleware* de IoT

Em uma representação simplificada, conforme a Figura 2.2, o *middleware* de IoT é uma camada de software entre a camada física (ambiente dos dispositivos, coleta de dados, processamento local de dados, execução de comandos com ações sobre o ambiente externo à IoT, dentre outros.) e a camada de aplicação (ambiente de tratamento consolidado dos dados, inferências, decisões, preparação de comandos, tarefas administrativas e coordenação da segurança). Como elemento de intermediação, o *middleware* proporciona um conjunto de abstrações, de modo a facilitar a integração e comunicação entre componentes heterogêneos [38].

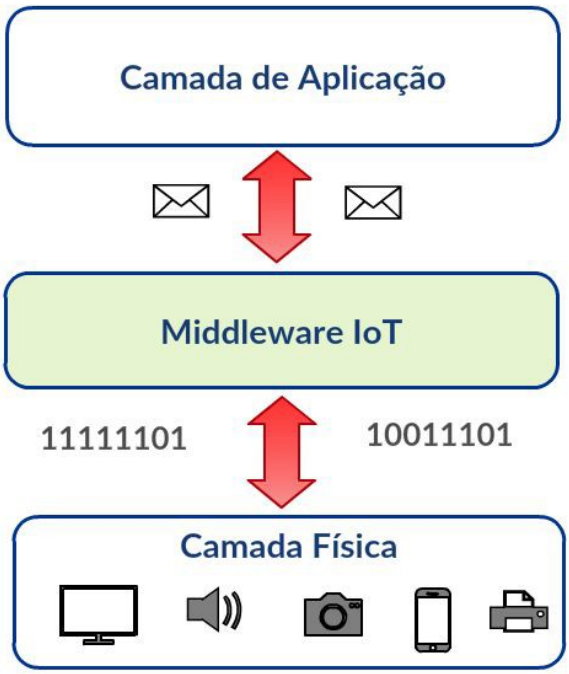


Figura 2.2: Estrutura básica do *middeware* de IoT [3]

O *middleware* abstrai as complexidades dos sistemas locais ou hardware dos dispositivos, permitindo ao desenvolvedor da aplicação focar todos seus esforços nas tarefas para serem resolvidas, sem os fardos do trabalho no nível dos dispositivos, tais como as complexidades relacionada às questões de comunicação. Desse modo, o objetivo do *middleware* é esconder os detalhes tecnológicos dos objetos físicos (coisas) e oferecer múltiplos serviços para os desenvolvedores de aplicações [39] [40]. Outro objetivo é proporcionar funções críticas comuns a todas as aplicações e que, portanto, ficam melhor estruturadas se fatoradas e colocadas no *middleware*, em vez

de serem repetidas em cada aplicação. Tais funções incluem: serviço de descoberta, gerenciamento de dados, controle de acesso, serviços de associação dispositivo-aplicação, abstrações de dados, entre outros [41].

Em uma representação mais estruturada da instância de IoT, é necessário considerar a presença de múltiplos dispositivos, assim como múltiplas aplicações inclusive aquelas de auto sustentação e administração da instância de IoT (ambiente de tratamento consolidado dos dados, inferências, decisões, preparação de comandos, tarefas administrativas, coordenação da segurança etc.), além da possibilidade de o *middleware* de IoT ser acoplado e usar serviços de outros módulos associados de suporte, como por exemplo *middleware* de serviços de dados e módulos locais de serviços de segurança da IoT, conforme descritos respectivamente em [36] e [10]. Por essa razão, na Figura 2.3 apresenta-se uma estrutura mais completa de um *middleware* experimental denominado UIoT [2] cujo desenvolvimento é prévio à presente tese. Este *middleware*de IoT é usado na validação da proposta do Modelo P2PIoT, onde esta validação é assunto do Capítulo 4.

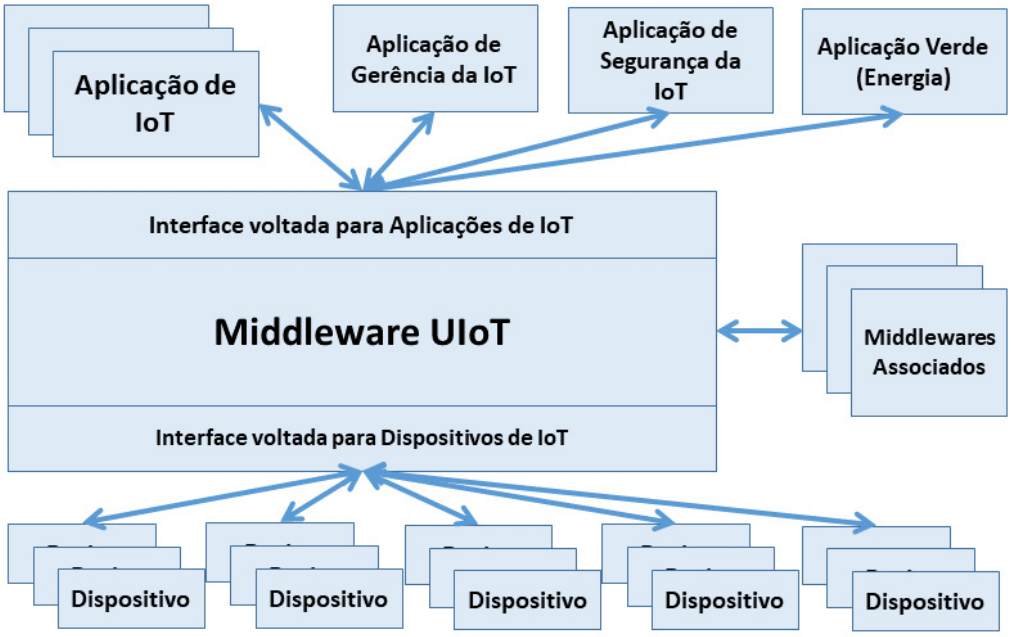


Figura 2.3: Estrutura do *Middleware* UIoT [2]

Considerando a Figura 2.3, o fluxo dos dados nos sistemas IoT atravessa as três camadas (aplicação, *middleware* de IoT e camada física) por meio do *middleware* de IoT que recebe as mensagens da camada física, executa processamento e encaminha para a aplicação e vice-versa, o que implica definir interfaces de serviços voltadas para a camada de aplicação e para a camada física.

A comunicação entre as diferentes camadas que compõem um sistema IoT normalmente é feita por meio de mensagens que podem ser do tipo requisição ou resposta. Uma mensagem de requisição dá a ordem para a execução de uma ação, e uma mensagem de resposta envia a resposta de uma requisição. O formato das mensagens depende de cada *middleware*. Uma possibilidade é usar o padrão JSON [42] [43] para lidar com a heterogeneidade dos dados geradas pela IoT. Por exemplo, a camada de aplicação origina uma mensagem JSON de requisição, para logo encaminhá-la ao middleware, até chegar na camada física da arquitetura IoT. A camada física executa a requisição. Após a execução, uma mensagem JSON é gerada como resposta e encaminhada para a camada de aplicação.

A característica de escalabilidade nos *middlewares* é um requisito essencial. Basicamente, a escalabilidade se refere à propriedade de um sistema aumentar a sua capacidade total diante do aumento de carga e à medida que recursos de hardware são adicionados, e utilizando protocolos de comunicação padrão e interoperáveis. Entretanto, há dificuldades importantes de padronização para um *middleware* escalonável e adaptável para computação pervasiva [39].

Já os *middlewares* existentes podem ser agrupados com base na arquitetura das abordagens [44], o que inclui as seguintes classes:

- *Middleware* baseado em eventos: os desta classe fazem uso das mensagens que são enviadas por aplicações (produtores) para serem recebidos por outras aplicações (consumidores). Tais mensagens são originadas a partir da ocorrência de eventos e as entidades usam um padrão de acesso a mensagens do tipo *publish*/*subscribe*;

- *Middleware* Orientado a serviços: tem por base a arquitetura orientada a serviços (*Service Oriented Architecture* – SOA), tipo de abordagem caracterizado pela neutralidade tecnológica, baixo acoplamento, reutilização, composição e descoberta de serviço. Tal abordagem pode dar suporte à implantação, publicação, descoberta e acesso ao serviço em tempo de execução;

- *Middleware* orientado a *Virtual Machine* – VM: provê suporte para a programação criando um ambiente seguro de execução para usuários de aplicações virtualizando a infraestrutura. Desta forma, as aplicações são divididas em pequenos módulos separados, os quais serão inseridos e distribuídos através da rede;

- *Middleware* orientado a Agentes: divide as aplicações em módulos para facilitar a distribuição através da rede usando agentes móveis. Esses mantêm seu estado de execução ao migrar de um nó para outro. Isto facilita a criação de sistemas descentralizados tolerantes a falhas [45];

- *Middleware* baseado em tupla-espaço: cada membro da infraestrutura mantém uma estrutura de tupla-espaço local. Uma tupla-espaço é um repositório de dados que pode ser acessado de forma concorrente [46]. O conjunto das tuplas-espaços forma uma federação. Essa abordagem é adequada para dispositivos móveis em uma infraestrutura de IoT, pois podem compartilhar dados de forma transitória dentro das restrições de conectividade [44].

- *Middleware* orientado a Banco de Dados: nestes uma rede IoT, por exemplo, uma rede de sensores, é vista como um sistema de banco de dados relacional virtual, com a camada de *middleware* atuando como um processador de consultas centralizado e cada objeto da rede possuindo seu processador de consulta que encaminha resultados ao *middleware*. Uma aplicação pode consultar essas informações usando uma linguagem de consulta similar ao SQL, o que permite elaborar consultas complexas [47].

- *Middleware* orientado a aplicação: uma abordagem específica de aplicação foca no suporte do gerenciamento de recursos para um destino específico ou para o domínio da aplicação, o que é possível graças à implementação de uma arquitetura ajustada a rede ou a uma infraestrutura com base nos requisitos da aplicação ou do domínio da aplicação.

Em todos os casos, o número indefinido, e que vem se apresentando sempre crescente, de dispositivos impõe um desafio particular quanto ao registro de dispositivos junto a qualquer das formas de *middleware*, haja vista a impossibilidade de conhecimento geral de todos os possíveis dispositivos. Por essa razão, uma possibilidade interessante é que o próprio dispositivo tome a iniciativa do registro conforme discutido a seguir.

2.3 Auto Registro de Dispositivos IoT

Tradicionalmente, na literatura, o gerenciamento de serviços em redes IoT é concebido em uma abordagem centralizadora, em que um conjunto de nós na rede tem a responsabilidade de tomar a iniciativa da descoberta dos dispositivos e serviços, além de realizar a gerência dos mesmos. Tal responsabilidade pode também ser atribuída a um servidor com alto poder de processamento e comunicação, por exemplo, uma nuvem computacional.

Ambos os cenários fundamentalmente dificultam a escalabilidade, haja visto que o gerenciamento com entidades centralizadoras esbarra no limite quanto à quantidade de recursos ativos disponíveis em um determinado momento. Ademais, tal forma de estruturação desconsidera o crescimento do poder computacional dos próprios dispositivos atuais, que são capazes de realizar diversos processamentos antes de propagar informações em uma rede.

Tendo em vista que um dispositivo pode fornecer um conjunto de serviços, e considerando os recursos computacionais presentes nos microcontroladores de hoje, em artigo precedente e próximo a esta tese [2], é proposta uma abordagem para que o próprio dispositivo possua o controle sobre os serviços que disponibiliza, e tome a iniciativa de atualizar a rede sobre novas informações relativas a cada um de seus serviços, visando aumentar a escalabilidade da rede, além de possibilitar um melhor controle sobre a quantidade de dados trafegados.

O auto registro permite ao *middleware* criar uma abstração computacional correspondente ao dispositivo, abstração esta em que são colocados os atributos do dispositivo, os serviços que o dispositivo provê e os parâmetros de entrada e saída para uso na ativação de tais serviços.

2.4 Conceituação de Interoperabilidade e sua Contextualização em IoT

2.4.1 Múltiplas Facetas da Interoperabilidade de Sistemas

A interoperabilidade é definida pelo Institute of Electrical and Electronics Engineers – IEEE [48] como "a habilidade de dois ou mais sistemas ou componentes trocarem informação e serem capazes de utilizar a informação trocada". Essa habilidade deve evitar algum tipo de esforço especial por parte de um cliente, o que deve ser possível tão somente pela aplicação de normas e do uso de padrões. Nesse sentido, a interoperabilidade está atrelada a cooperação, normalizada por especificações, políticas e padrões que viabilizem o intercâmbio integrado de informações. Entretanto, haja vista que a conceituação de interoperabilidade não é consensual, várias outras definições também vinculadas ao IEEE indicam a variedade de significados do termo (IEEE, 2000):

- A habilidade de dois ou mais sistemas trocarem informações entre si e fazer uso dessas informações trocadas.
- A capacidade de unidades de equipamentos trabalharem conjuntamente na realização de funções úteis.
- A capacidade promovida, mas não garantida, de aderir a um determinado conjunto de padrões, o que possibilita a distintos equipamentos trabalharem em rede.
- A habilidade de dois ou mais sistemas ou componentes trocarem informações em uma rede heterogênea e usar essas informações.

Tais definições implicam na classificação da interoperabilidade entre sintática e semântica: a primeira se refere à interoperabilidade ao nível de mensagem (por exemplo, o formato e a sequência de campos de dados) e a segunda se refere à interoperabilidade no nível do significado dos dados (por exemplo, a identificação de um campo específico dentro do documento trocado, bem como a faixa de valores e a unidade de grandeza). Desse modo, o dado pode não apenas ser transmitido pela rede, como também ser processado automaticamente pelo receptor, uma vez que existe entendimento quanto ao significado dos dados trocados entre os sistemas (ou seja, se mantém o significado original da informação).

O conceito de interoperabilidade é também referenciado em computação como a capacidade de sistemas operacionais operarem e cooperarem mesmo na presença de diferentes representações de dados e protocolos de comunicação [49]. Por extensão, uma conceituação mercadológica apresenta a interoperabilidade como a capacidade do software e do hardware, pertencentes a diferentes máquinas e de diferentes marcas comerciais, compartilharem dados [50].

No âmbito da Ciência da Informação e campos correlatos da Biblioteconomia e Documentação, a interoperabilidade, de maneira geral, é entendida como a capacidade que os sistemas de

hardware e software têm para se comunicar e operar com outros sistemas no intercâmbio de dados [51] [52]. Há referência assim ao fluxo da informação e aos ciclo da informação, distiguindo os tempos ou etapas da produção da informação, da comunicação e do uso da informação.

Outro perspectiva [53] diz respeito a interoperabilidade como a possibilidade do usuário buscar por recursos informacionais heterogêneos, armazenados em diferentes locais de uma rede, utilizando-se de uma interface única e sem necessidade de conhecimento sobre como os recursos estão armazenados. Assim, a interoperabilidade é resolvida com serviços úteis para os usuários, a partir de recursos informacionais que são tecnicamente diversos e muitas vezes gerenciados por instituições diferentes, o que leva à questão das fontes de informação manipuladas pelos sistemas, as quais podem apresentar diferenças sintáticas, estruturais ou semânticas que constituem os fatores de interpretações divergentes para a informação intercambiada entre os sistemas [54].

Segundo [55], para mitigar esses diferentes tipos de problemas, a cooperação entre sistemas deve ser promovida em três níveis:

- Acordo técnico: que busca promover interoperabilidade tecnológica, através da uniformidade da informação e dos serviços utilizados entre os diferentes sistemas, o que abarca formatos, protocolos e padrões, de forma que mensagens possam ser trocadas entre tais sistemas;
- Acordo sobre conteúdo: voltado à interoperabilidade semântica, utilizando a representação e organização do conhecimento, com o uso de metadados e recursos para uniformizar a interpretação de mensagens;
- Acordo organizacional: que trata de regras básicas para acesso, alteração e autenticação da informação, bem com integração entre serviços, buscando reduzir diferenças políticas, através da reunião das organizações em acordos federativos com o intuito de implementar padrões e tecnologias comuns.

Outros pontos de vistas similares desenvolvem mais amplamente o conceito de interoperação, incluindo capacidade de comunicação, de troca de informações, de uso de operações mutuamente, de forma independente das arquiteturas, plataformas e semânticas utilizadas, em consideração a variáveis contextuais. Nessa visão mais abrangente, há outros fatores determinantes da interoperabilidade, incluindo:

- Interoperabilidade técnica: padrões de comunicação, de transporte, de armazenamento e de representação da informação;
- Interoperabilidade semântica: significado da informação originada em diferentes sistemas, com providências e recursos para assegurar interpretações uniformes entre os sistemas, o que inclui definição de metadados, classificação, tesauros e ontologias;
- Interoperabilidade organizacional: relacionada ao contexto organizacional e processos de negócio, ou seja, incluindo fluxos de trabalho e de informação, relações de poder e cultura da instituição;

- Interoperabilidade política e humana: envolve a forma como a informação é disseminada, sua classificação e a decisão de disponibilizar na organização;
- Interoperabilidade intercomunitária: aborda o acesso a informações originadas em diferentes fontes, por organizações, especialistas e comunidades de natureza distintas; remete à interação entre domínios independentes;
- Interoperabilidade legal: relacionada a exigências e implicações legais de tornar a informação livre e amplamente disponível;
- Interoperabilidade internacional: envolve a cooperação em escala internacional, onde o intercâmbio envolve uma grande diversidade de padrões e normas, além de problemas inerentes de comunicação por barreiras linguísticas.

Ainda, nesse sentido, entende-se também a interoperabilidade como resultante de processos, tecnologias e protocolos requeridos para assegurar a integridade dos dados, quando se transferem de um sistema a outro, assim como a transmissão de resultados consistentes e com significado para o usuário final[56].

Portanto, na definição de interoperabilidade, insere-se a adequada interconexão de sistemas e o intercâmbio de dados, informação e conhecimento entre eles. Nesse sentido, o Modelo proposto nesta tese para interoperação de instâncias heterogêneas de IoT é descrito em termos de componentes estruturais a serem integrados aos *middleware* de controle de cada instância, bem como em termos de funcionalidades de serviços e de abstração de dados que possam ser usados indistintamente em todas as instâncias de IoT que devam interoperar.

O desenvolvimento de trabalhos correlatos nesse mesmo contexto é discutido na seção a seguir, de modo a caracterizar as contribuições da proposta desta tese com relação a tais trabalhos.

2.4.2 Trabalhos Correlacionados sobre Interoperação em IoT

Muitos trabalhos sobre IoT abordam o tema da estrutura do *middleware* e sua disponibilidade para integrar múltiplas instâncias de IoT, em particular por meio da utilização de nuvem [24] [37]. Esse tipo de solução centralizada é importante para endereçar a questão da disponibilidade do *middleware* e seus componentes, ainda que não tenha foco sobre a questão da interoperação entre diferentes *middlewares*.

Na referência [57] encontra-se a proposição de uma estrutura conceitual modularizada em relação aos papéis dos elementos de uma infraestrutura de IoT que são especificados no vocabulário e nas regras de uma linguagem descritiva específica, o que permite tanto modelar dispositivos existentes quanto gerar códigos executáveis para utilização pelos aplicativos.

Com base em [57], a proposta de [58] introduz estruturas de dados para resolver problemas de heterogeneidade em IoT por intermédio de componentes da lógica do aplicativo (*Application Logic* - AL) e de entidades de serviços comuns (*Common Service Entities* - CSE), sendo os citados

componentes AL destinados a abstrair os mecanismos do aplicativo voltados para os usuários, como interface do usuário, mecanismos de consulta e descoberta de recursos, entre outros. Já as CSE abstraem os serviços disponíveis para os aplicativos, incluindo a dinâmica de registro, o gerenciamento e a interoperabilidade com os dispositivos.

Com o mesmo objetivo, a referência [59] propõe um arcabouço de três camadas para resolver a questão da heterogeneidade de dispositivos e protocolos em IoT, sendo uma camada destinada a gerenciar a dependência relacionada as protocolo dos dispositivos, enquanto as duas outras integram abstrações que permitem a comunicação e a colaboração com os dispositivos constituintes de uma IoT.

Já em [60], a proposta é bastante diferente das supracitadas, no sentido de que compõe modelos e *scripts* de tarefas numa visão de usuário, de modo a reduzir o uso da programação em níveis mais próximos do hardware.

Um trabalho anterior [61] diretamente relacionado a esta tese propõe uma API para acessar serviços baseados em IoT através dos protocolos REST e SOAP, de modo a fornecer meios para tratar da complexidade inerente à diversidade de dispositivos que servem à construção de aplicativos IoT. Nessa proposta, uma estrutura de *middleware* fornece acesso uniforme a recursos e operações por meio de mensagens estruturadas em SOAP e REST que são especificadas tendo em vista a utilização do padrão UPnP para registro e abstração das capacidades dos dispositivos de IoT.

Do mesmo modo, o trabalho [62] objetiva um modelo de representação uniforme dos dispositivos, por intermédio de uma plataforma de configuração usada primeiro para o desenvolvimento de novos aplicativos e depois para que tais aplicativos usem serviços baseados em IoT. Isso é feito com a utilização de XMPP nas comunicações entre entidade e em uma API de virtualização.

Também explorando a virtualização, a proposta de [63] segue os princípios da proposta de [61] argumentando que uma virtualização globalizada e única tem o potencial de reduzir a complexidade inerente ao desenvolvimento de aplicativos de IoT, facilitando a codificação quanto ao tratamento dos diversos dispositivos existentes.

Tais abordagens usam formas de abstrair dispositivos, deixando as características de interoperação sob responsabilidade dos modelos de aplicativos e em geral deixando alguns serviços comuns para trabalhos futuros ou complementares. Por exemplo, há trabalhos que citam a necessidade de mecanismos de descoberta de rede, como [58], sem entrar em detalhes de especificação. Já [61] apresenta uma ideia de auto-registro de dispositivos junto ao *middleware*. As ideias de virtualização em geral são exploradas com protocolos específicos, como [62].

Vale notar que os trabalhos mencionados são principalmente dedicados a aspectos de interoperabilidade internos às instâncias de IoT. Como esta tese está focada em questões de interoperabilidade entre diferentes instâncias de IoT, nossa proposta lida com recursos e serviços de interoperabilidade entre essas diferentes instâncias.

Nossa proposta pressupõe que os meios de interoperabilidade internos às instâncias de IoT

estão em vigor e podem ser mapeados pelos módulos propostos nesta tese para serviços de interoperação envolvendo instâncias de IoT heterogêneas que podem ser federadas, pelo menos temporariamente, por comando administrativo ou automaticamente em função de detecção de instâncias de IoT no mesmo espaço físico ou lógico.

Conforme já mencionado, esta tese também está correlacionada ao trabalho [2], onde se encontra uma abordagem de auto registro de serviços de dispositivos em ambientes de IoT, com a ideia de que o próprio dispositivo deve ser capaz de registrar seus serviços na instância de rede IoT, portanto de maneira distribuída. Entretanto, trata-se aí de questão de interoperabilidade interna à instância, mas é importante observar que o auto registro resulta na criação de abstrações dos dispositivos que são fundamentais para os os serviços entre diferentes instâncias IoT, no sentido de prover operações de mesclagem e divisão, com trânsito de dispositivos e dados entre tais instâncias.

Outro trabalho correlacionado precedente [64] trata da interoperação entre o *middleware* e redundantes *gateways IoT*, sendo cada *gateway*, na verdade, um *middleware* compacto. Nesse caso, uma operação distribuída entre pares similares é introduzida, ainda que cada *gateway* seja um mediador que espelha as operações do *middleware* de referência, sem oferecer operação totalmente distribuída entre *gateways* pares.

Já a referência [38] propõe um modelo de arquitetura de referência para *middleware IoT*, detalhando o método de operação de cada módulo proposto, com modelos de comunicação de dispositivos a aplicações, porém sem abordar a interoperabilidade entre *middlewares* heterogêneos, nem descrever as funcionalidade e serviços que são necessárias.

O trabalho [65] trata de um serviço de rede P2P em nuvem para informações de situações de desastre baseadas em IoT, em que a interoperação fica fora da IoT, sendo fornecida como Social Networking Service (SNS). Assim, esse serviço não trata de compartilhamento de dispositivos, serviços e interoperação de *middlewares* de diferentes instâncias de IoT utilizando uma rede P2P.

Já de acordo com [13], o conceito de federação entre dispositivos e sistemas atraiu um grande interesse nas comunidades de pesquisa de IoT para promover interconectividade entre plataformas de IoT com base em federação, unificação e interoperabilidade semântica entre vários domínios de IoT. No entanto, esse mesmo artigo destaca que as soluções que dependem exclusivamente da adoção de infraestruturas e serviços em nuvem provavelmente não conseguirão acompanhar os problemas esperados de escalabilidade e congestionamento de rede em futuros cenários de IoT, considerando os potenciais da computação na borda e nos dispositivos (*egde, fog*).

Como já exposto na Seção 1, na presente tese a proposta é prover um serviço de busca P2P acoplado com serviços de interoperação entre instâncias de IoT, de forma independente da escolha de um ambiente de nuvem ou de computação na borda, ou mesmo de um ambiente misto, para implantação dos módulos de cada instância IoT.

Diversos avanços tecnológicos nas últimas décadas permitiram a emergência do paradigma de IoT. Contudo, muitas questões desafiadoras ainda precisam ser abordadas e vários obstáculos tecnológicos ainda precisam ser suplantados para a concretização e a ampla utilização desse

paradigma. Tais desafios, anteriormente mencionados, vão desde soluções que permitam a interoperação e a integração dos diversos componentes que compõem os ambientes de IoT até a facilitação do desenvolvimento de aplicações para esses ambientes, passando por questões relativas à escalabilidade por conta do grande número de objetos (coisas) envolvidos e do grande número de eventos que podem ser gerados espontaneamente por esses mesmos objetos [66].

A esse propósito, uma recente referência [8] apresenta um breve levantamento das propostas de uso do *blockchain* como sistema de suporte P2P para IoT, discutindo os pontos de convergência de *blockchain* e IoT e apresentando uma proposta de arquitetura correspondente. Essa proposta não é desenvolvida em detalhes estruturais e funcionais, com a descrição de algoritmos, protocolos e abstrações de dados, o artigo considerando estes temas para um estudo mais aprofundado, haja vista uma série de desafios para a utilização de *blockchain* em IoT também discutidos no citado artigo.

Já na proposta desta tese, além da concepção geral do modelo, fazemos o detalhamento dos módulos de software que devem ser inseridos e adaptados nos diferentes *middlewares* e que são acoplados com um serviço de busca P2P que tem grande plasticidade no que se refere a aceitar novos recursos de maneira balanceada e com funcionalidades de estabilização e manutenção da integridade referencial, aspectos que garantem a escalabilidade. Tal propriedade é inclusive verificada no processo de validação da proposta da tese por intermédio de testes de um protótipo desenvolvido segundo o modelo proposto.

No sentido de antecipar as funcionalidades de interoperação do Modelo proposto, listamos desde já os serviços vislumbrados para a interoperação entre instâncias heterogêneas de IoT segundo tal modelo:

- Agrupar e desagrupar instâncias de IoT
- Substituir dispositivos com recuperação da base informacional completa;
- Copiar a base informacional de um dispositivo para outro;
- Consolidar bases informacionais de vários dispositivos, para agregação de conhecimento;
- Duplicar dispositivos de uma instância para outra;
- Transportar dispositivo de uma instância de IoT para outra;
- Fazer um dispositivo participar em duas instâncias diferentes de IoT, com entrada e saída dinâmica.

Na seção seguinte, são discutidas as características do sistema de busca P2P que é requerido na proposta P2PIoT desta tese para implementar tais serviços.

2.5 Sistemas de busca P2P e Paradigma Chord

O conceito de sistema entre pares (peer-to-peer – P2P), assim como os correspondentes protocolos, algoritmos e estruturas de dados, se contrapõe a outros paradigmas existentes, à medida que não depende de uma organização central ou hierárquica, além de atribuir às entidades integrantes as mesmas capacidades e responsabilidades [67].

Desse modo, qualquer das entidades pares tem a possibilidade de acessar diretamente os recursos de outra entidade, sem nenhum controle centralizado ou hierárquico, ou seja, o sistema P2P se encarrega de colocar o par de entidades em cooperação direta, obviamente devendo levar em conta outros aspectos além das funcionalidades de cooperação, tais como aspectos administrativos e contratuais, segurança (em especial: autenticação, autorização, auditabilidade), requisitos de desempenho e outros.

A inexistência de um servidor central significa que é possível cooperar para a formação de uma rede P2P sem qualquer investimento adicional em hardware de alto desempenho para coordená-la. Outra vantagem é a possibilidade de agregar e utilizar a capacidade de processamento e armazenamento que fica subutilizada em máquinas ociosas.

Além disso, a natureza descentralizada e distribuída dos sistemas P2P torna-os inerentemente robustos a certos tipos de falhas muito comuns em sistemas centralizados.

Finalmente, o modelo P2P apresenta características de algoritmo e protocolos que contribuem para a escalabilidade, por apresentar plasticidade ou maleabilidade no que se refere a tratar do crescimento indefinido, *a priori*, no número de usuários e equipamentos conectados, tendo em conta as capacidades adaptativas de rede, aplicações e capacidade de processamento.

Para descrever de forma mais detalhada tais características, tomamos a seguir o Modelo Chord [4], considerado em geral um paradigma de sistema P2P. Conforme já apresentado, por ter código fonte disponível para a pesquisa, tal modelo é usado na fase de validação da proposta desta tese.

2.5.1 Modelo Chord

Chord [4] é um algoritmo criado pelo Massachusetts Institute of Technology – MIT, com a concepção de um sistema para armazenamento de grandes quantidades de dados divididos de forma balanceada entre as entidades pares, ou seja, havendo um volume total D de dados a serem divididos entre N entidades pares, cada entidade par armazena uma parte D/N do volume de dados.

Tais entidades mantém uma estrutura de roteamento e algoritmos que permitem fazer a busca dos dados com características de desempenho eficientes e adaptáveis.

Entende-se por eficiência em Chord que o tempo para encontrar os dados é logarítmico, assim como é o tamanho das tabelas de roteamento, ou seja, para poder encontrar D itens de dados, basta possuir $\log_2(D)$ ponteiros na tabela de roteamento. Por exemplo, para $1.000.000$ de

itens bastam $\log_2(1000000) \approx 20$ ponteiros.

Já a adaptabilidade é entendida no sentido de que, aumentando linearmente o número de informações de roteamento (ponteiros), diminui-se o tempo de busca exponencialmente, ou seja, acrescentando 1 ponteiro à tabela de roteamento, reduz-se pela metade o tempo de roteamento. No pior caso, havendo N nodos e apenas 1 ponteiro, o tempo de busca é de N unidades de tempo. Com P ponteiros adicionais, o tempo de busca cai para $N/2^P$.

Para tanto, a disposição informacional em Chord consiste em atribuir identificadores aos nodos P2P (e aos conteúdos armazenados) com tais identificadores sendo ordenados em um anel de números inteiros, com um sentido horário de crescimento.

Além de garantir que os dados dispostos no anel possam ser acessados de forma ágil, Chord possui um método e protocolo para garantir que entradas ou saídas de nodos do anel causem um impacto mínimo no desempenho, no sentido de que somente sejam afetados dados do nodo sainte ou entrante e que a estabilidade do anel seja mantida, em particular pelo fato de ser necessário mover tão somente D/N itens quando cada nodo entra ou sai (ou falha).

O efeito emergente de tais propriedades é a escalabilidade do Chord, com estabilização diante de eventos de entradas e saídas (estas inclusive por falhas) dos nodos.

A Figura 2.4 mostra uma representação comum para um anel de identificadores Chord. A quantidade de identificadores é 2^H, sendo H a quantidade de bits usada para os resultados do algoritmo de *hash* que é usado na atribuição de identificadores, conforme explicado adiante. No exemplo da figura 2.4, há $2^3 = 8$ identificadores disponíveis no anel, numerados e na ordem crescente no sentido horário de 0 a 7. Os identificadores são atribuídos tanto para nodos, que são as entidades pares interoperantes no anel, quanto para os itens armazenados pelos nodos. As regras de atribuição para os nodos e para os itens armazenados são discutidas em sequência a seguir.

No que se refere aos nodos, no exemplo da Figura 2.4, há 3 nodos representados por bolas verdes, usando os identificadores 0, 1 e 3 disponíveis no anel mas não necessariamente consecutivos, haja vista que a atribuição dos identificadores é produto do algoritmo de *hash* que espalha os nodos aleatoriamente no anel.

O funcionamento do Chord segue a ideia de que cada nodo, ao ser inicializado, possui um sucessor e um antecessor. Quando o nodo é adicionado a um anel que já contenha outros nodos, o nodo entrante irá realizar um cálculo para definir onde deverá estar posicionado dentro do anel, ou seja, para ter a si atribuído um dos identificadores disponíveis no anel. Este cálculo é feito com uma função de *hash* como o SHA1 ou MD5.

Uma vez calculada esta posição, o nodo nela se insere, informando aos outros nodos e se colocando como um sucessor e antecessor de um par dos outros nodos. Os dois outros nodos envolvidos também atualizam a suas informações, um deles sobre o novo sucessor e o outro sobre o novo antecessor, de modo que é mantida uma dupla cadeia de ponteiros nos nodos configurando o círculo dos nodos pares. Esse procedimento é denominado de *merge*.

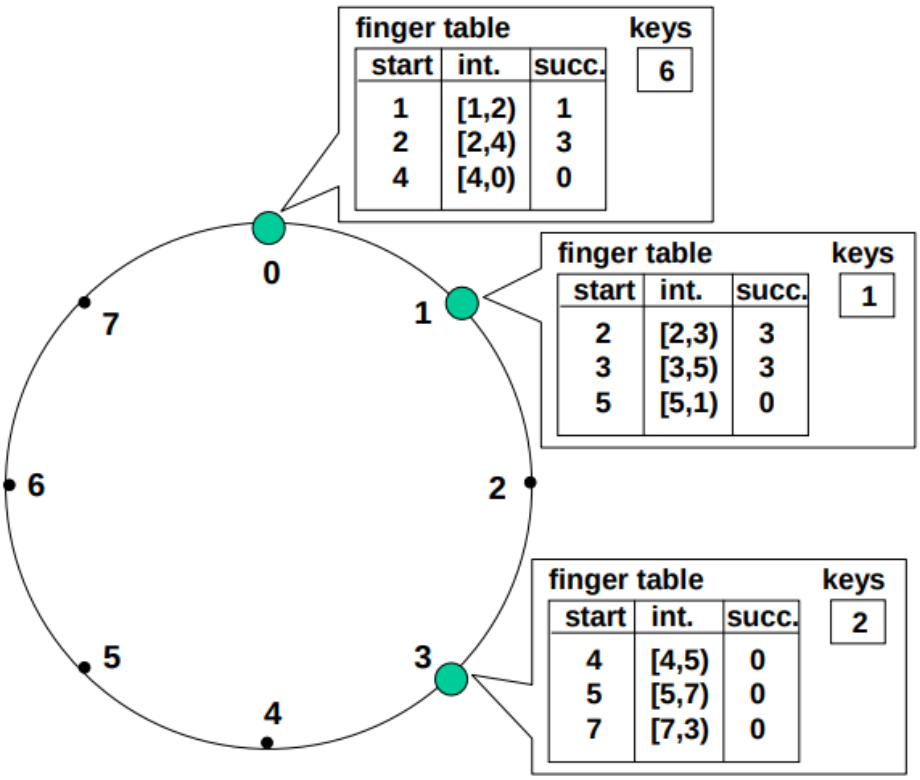


Figura 2.4: Representação de um anel chord [4]

Assim, reconsiderando a Figura 2.4, há um anel formado por uma função de *hash* de $H = 3$ bits, ou seja, $2^H = 8$ identificadores representados como um anel de números de 0 a $2^H - 1$. Nesse anel, os identificadores 0, 1, e 3 foram atribuídos a nodos reais que estão marcados por bolas de cor verde, apenas esses nodos fazendo parte da rede P2P.

No círculo de nodos, considerando um determinado nodo k, o $sucessor(k)$ é definido como sendo o identificador do primeiro nodo real seguinte a k, no sentido horário do anel. Assim considerando os nodos 0, 1, e 3 da Figura 2.4, tem-se $sucessor(0) = 1$, $sucessor(1) = 3$ e $sucessor(3) = 0$. Já os antecessores, são definidos na ordem inversa, ou seja, no sentido anti-horário do percurso do círculo de nodos. Tais informações de sucessores e antecessores são armazenadas na tabela de ponteiros (*finger table*) em cada nodo.

Já no que se refere aos itens armazenados (que podem ser quaisquer tipos de arquivos de áudio, vídeo, imagem, documento, programa, ou seja, qualquer tipo de arquivo armazenável), os nomes dos itens também são mapeados para identificadores Chord através da função de *hash* escolhida, sendo tais identificadores denominados chaves. Para converter o nome do item (que será um arquivo a ser procurado no sistema P2P) na respectiva chave usa-se a função $chave = hash(nome)$. Para que os nomes sejam unívocos ao tornar os itens disponíveis no Chord, primeiro se deve criar uma tupla de indexação do tipo (nome, endereço IP) e depois calcular $sucessor(hash(nome))$, identificado desse modo o nodo Chord ao qual será pedido para

armazenar a citada tupla. Assim, pela distribuição aleatória operada pela função *hash* os itens serão distribuídos na proporção D/N no círculo de nodos Chord.

Para mais tarde realizar a busca do item pelo nome, será necessário calcular $chave = hash(nome)$ e em seguida com tal $chave$ calcular $sucessor(chave)$ para então encontrar nesse sucessor o endereço IP do nó que efetivamente armazena o item, tal como identificado na correspondente tupla de indexação [68]. Para essa pesquisa de itens, há em cada nodo a já citada tabela de ponteiros *finger table*, na qual se encontram os pares $chave - valor$ que identificam os itens indexados pelo nodo.

Como as informações de indexação estão dispostas em vários nodos, o círculo de nodos na verdade constitui uma tabela distribuída de *hashes* (*Distributed Hash Table* – DHT), resultando em uma configuração P2P flexível, estável, adaptável e escalonável para buscas eficientes de dados em sistemas heterogêneos distribuídos.

2.5.2 *Distributed Hash Table* em Chord

Outra maneira de descrever o Chord consiste justamente em considerar o conceito de DHT e exemplificar a partir desse conceito, conforme [5].

Nesse sentido, inclusive para trazer à baila um exemplo mais próximo da temática desta tese, nos referimos ao caso ilustrado na Figura 2.5 em que se apresenta uma DHT contendo uma relação de dispositivos de IoT e o respectivo *middleware* responsável por cada dispositivo.

Vê-se nessa figura 2.5 que a DHT é uma *hash table* comum organizada em tuplas chave-valor, mas que tem a especificidade de ter partes que são distribuídas entre vários nodos computacionais, cada um com seu próprio endereço IP unívoco. Cada nodo fornece uma operação de pesquisa ou busca (*lookup*) que retorna o valor associado a uma chave. Como estão distribuídos, os nodos mantêm ponteiros de roteamento, de modo que, se o item buscado não for encontrado em determinado nodo, este direciona a busca para outro nodo do conjunto.

Para a construção dessa DHT em Chord, usa-se um espaço de nomes lógico, chamado de espaço de identificadores, consistindo dos identificadores $0, 1, 2, \ldots, N-1$. O espaço de identificadores é um anel lógico $mduloN$ e cada nodo pega um identificador randômico nesse espaço, conforme apresentado na Figura 2.6(a). Então, com base no conceito de sucessor do identificador, cada nodo tem um sucessor que é primeiro nodo encontrado na direção horária do anel, conforme Figura 2.6(b). Há também o conceito de predecessor do nodo, não mostrado na Figura, que é o primeiro nodo encontrado na direção anti-horária do anel.

Para armazenar dados na DHT Chord, usa-se uma função de *hash* globalmente conhecida, $H()$. Para cada item $< chave, valor >$ é calculado o identificador $k = H(chave)$. Então, armazena-se cada item k no nodo sucessor de k entre os nodos presentes, ou seja, tal nodo é responsável pelo item k, conforme Figura 2.6(c);

Para garantir consultas rápidas, bem como flexibilidade na entrada e saída de nodos, cada

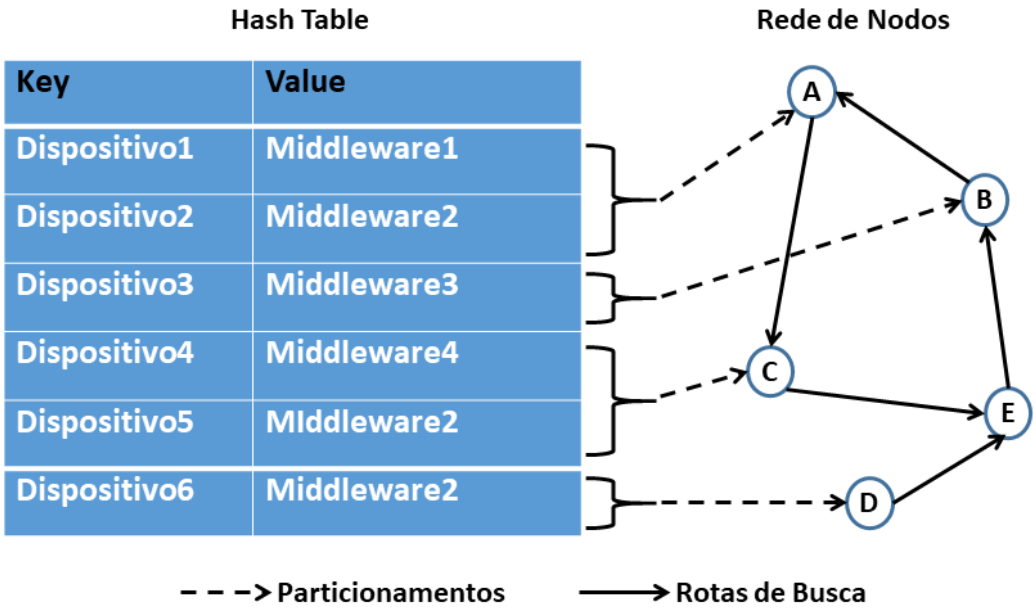


Figura 2.5: Exemplo de DHT para associar dispositivos de IoT com o *middleware* de controle de uma instância de IoT. Adaptado de [5].

nodo Chord mantém ponteiros em uma tabela de rotas (*finger table*). Isso permite que cada nodo aponte para seu sucessor, sendo designado o sucessor de um nodo n como $succ(n+1)$, conhecido como ponteiro $succ$ do nodo, conforme mostra a Figura 2.7(a). Não é mostrado na Figura, mas cada nodo aponta também para o seu predecessor que é o primeiro nodo encontrado no sentido anti-horário a partir de $n-1$, conhecido como ponteiro $pred$ de um nodo.

Tal configuração, permite realizar a operação de consulta/pesquisa/busca (*lookup*) no Chord. Nesta operação, para procurar uma chave k, calcula-se $H(k)$ em qualquer nodo em que a consulta seja disparada. A partir desse nodo, segue-se então os ponteiros $succ$ consultando cada nodo até que o item k seja encontrado, conforme mostram as Figuras 2.7(b) e 2.7(c).

Para acelerar a operação de consulta/pesquisa/busca (*lookup*) no Chord, leva-se em conta que se somente o ponteiro para $succ(n+1)$ for usado em cada nodo, então o pior tempo de busca é N unidades de tempo, para um sistema com N nodos presentes. Para melhorar o tempo de pesquisa, usam-se mais ponteiros em cada nodo, conforme mostra a Figura 2.8: ponteiro para $succ(n+1)$, ponteiro para $succ(n+2)$, ponteiro para $succ(n+4)$, ponteiro para $succ(n+8)\ldots$ ponteiro para $succ(n+2M)$. Com cada ponteiro adicional, a distância ao destino é sempre reduzida pela metade. Assim, graças a um aumento linear de custo de manter ponteiros, obtém-se um ganho exponencial no desempenho da busca.

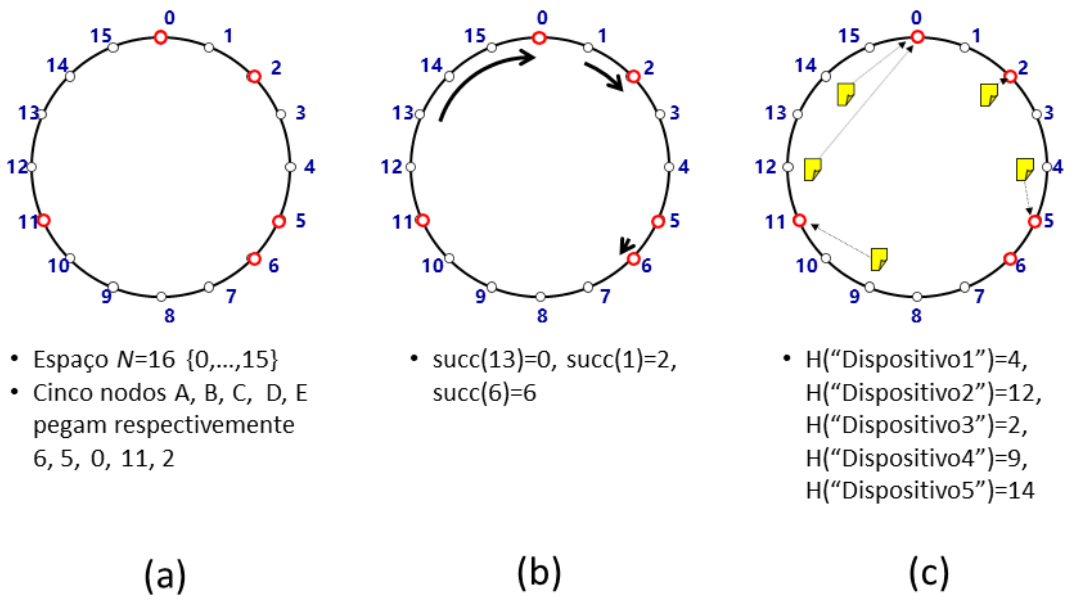


Figura 2.6: Construção de uma DHT em chord e armazenamento de itens nessa DHT. Adaptado de [5].

Essa configuração permite ainda lidar com o dinamismo no Chord, causado por falhas, bem como entrada e saída de nodos do anel.

Para lidar com falhas, cada nodo mantém uma lista de sucessores, ou seja, ponteiros para f sucessores mais próximos: $succ(n+1)$, $succ(succ(n+1)+1)$, $succ(succ(succ(n+1)+1)+1)$... Assim, se o sucessor falhar, será substituído pelo sucessor vivo mais próximo. Já se o antecessor falhar, define-se $pred$ para o valor nil e aguarda-se o processo de estabilização periódica do anel.

A estabilização periódica é usada para fazer os ponteiros eventualmente convergir para um correto duplo encadeamento no anel. Esse processo basicamente tenta apontar $succ$ para o sucessor vivo mais próximo e tenta apontar $pred$ para o predecessor vivo mais próximo, até verificar a correção do encadeamento duplo dos ponteiros.

Já para tratar a entrada de nodos, quando n se junta ao anel, encontra o sucessor de n com a operação $lookup(n)$ e define então $succ$ para o sucessor de n. A estabilização periódica corrige o resto do encadeamento de ponteiros no anel.

Já a saída de nodos é mais simples, pois quando n sai, apenas simplesmente desaparece (como numa falha) e a estabilização periódica recompõe o encadeamento de ponteiros do anel.

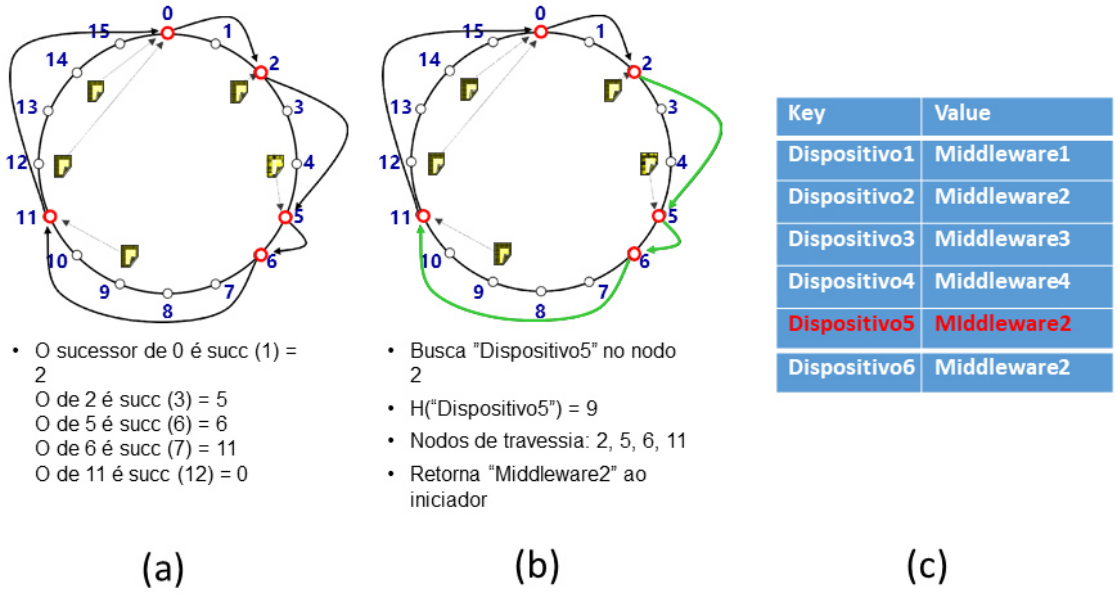


Figura 2.7: Consulta/pesquisa/busca (*Lookup*) na DHT em Chord. Adaptado de [5].

2.5.3 Rede de Sobreposição Chord

Com a estruturação de uma DHT, e a operação dos algoritmos Chord, o efeito emergente é uma rede de sobreposição.

Tal expressão é definida porque na rede de sobreposição os nodos, na verdade, enviam mensagens uns aos outros através de uma rede existente, como a Internet, que a rede de sobreposição usa como suporte à sua própria funcionalidade de roteamento, conforme ilustra a Figura 2.9.

A rede existente é então designada como rede subjacente, valendo notar que, se a rede subjacente for a Internet, quando a sobreposição encaminha solicitações entre os nodos da DHT então cada uma das correspondentes mensagens passa pelos roteadores e *switches* que formam essa rede subjacente.

Em uma rede de sobreposição, dada a rede subjacente sobre a qual a rede de sobreposição é construída, as mensagens entre os nodos na rede de sobreposição seguem logicamente a topologia de anel da rede de sobreposição, mas passam fisicamente pelas rotas formadas por enlaces e roteadores presentes na rede subjacente.

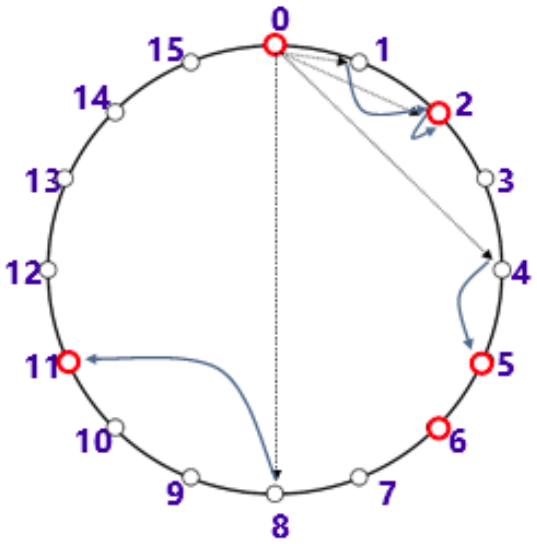


Figura 2.8: Aceleração da Consulta/pesquisa/busca (*Lookup*) na DHT em chord. Adaptado de [5].

2.6 Possível Convergência entre IoT e P2P

Há um interessante ponto de convergência entre o conceito de Instâncias de IoT e o de rede de sobreposição P2P, este especialmente no sistema Chord.

De fato, conforme ilustra a Figura 2.10, plano inferior, as instâncias de IoT têm uma estrutura física que envolve comunicações por intermédio da Internet, o que permite organizar os fluxos de informação entre dispositivos e computadores onde se encontram o *middleware* e as aplicações, com o tráfego veiculado pelos roteadores. Em várias situações *middleware* e aplicações compartilham a mesma plataforma (ilustrada por ícones sobrepostos de computadores) e em outras situações o *middleware* é replicado com unidades mais compactas, para atender a configurações específicas de grupos de dispositivos com limitações computacionais ou de protocolos de rede.

A organização dos fluxos de informação é atribuição do *middlware* que registra os dispositivos, gerencia abstrações de serviços destes dispositivos e vincula tais abstrações às aplicações. Desse modo se abstrairmos a estrutura física da rede e pensarmos tão somente em termos de fluxos de informação de IoT sob responsabilidade do *middleware*, então coloca-se por analogia que cada instância de IoT constitui uma sobreposição particular sobre a Internet subjacente, conforme ilustra a Figura 2.10, plano intermediário.

Com tal conceito de sobreposição, a convergência com o mesmo conceito em P2P, especialmente em Chord, é bastante imediata, haja vista que o *middleware* de cada instância de IoT pode constituir um nodo par na sobreposição Chord, conforme ilustra a Figura 2.10, plano superior, e dessa forma acessar os meios e serviços de interoperação com outras instâncias de IoT.

Considerando essa convergência conceitual e considerando as funcionalidades do sistema Chord e sua plasticidade para operar sobre a Internet, havendo além disso o código fonte do cor-

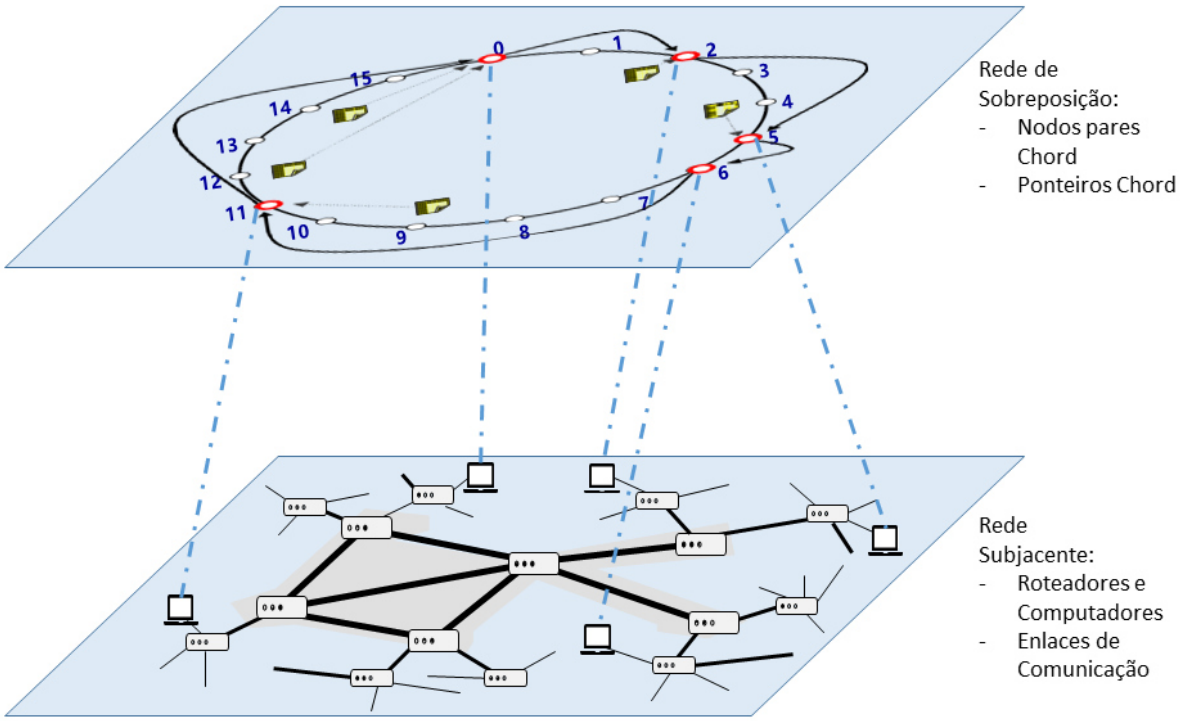


Figura 2.9: Rede de sobreposição Chord. Adaptado de [5].

respondente conjunto de funções que se encontra disponível para a pesquisa, tal sistema serve não apenas de inspiração para a construção da estrutura modular e dos serviços do Modelo P2PIoT proposto nesta tese, como também para o desenvolvimento do protótipo usado na validação da proposta, temas tratados respectivamente nos Capítulos 3 e 4 a seguir.

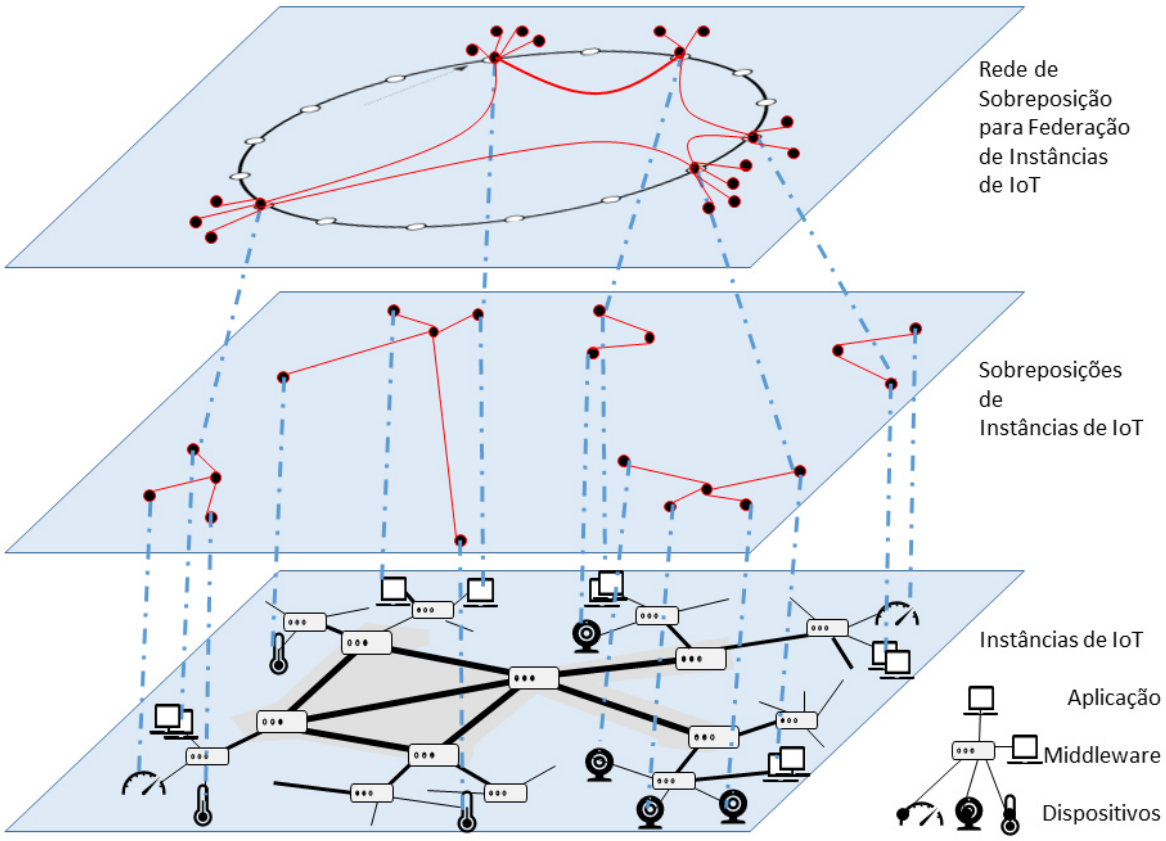


Figura 2.10: Convergência entre IoT e P2P/Chord.

Capítulo 3

Proposta de um Modelo P2P de Interoperação entre Instâncias de Redes IoT – P2PIoT

Argumentamos inicialmente no que se refere a uma definição de Internet das Coisas que, ao invés de ser uma rede única onde todos os dispositivos e todas as aplicações podem interagir, é mais realista referir-se ao conceito de que existem várias instâncias de redes IoT em execução paralela e simultânea.

Tais instâncias independentes se configuram também em razão de separação espacial ou de domínio administrativo, mesmo quando se considera a utilização de um conjunto de hardware e software homogêneo. Mas, com efeito, o caso geral de IoT envolve plataformas de hardware e sistemas operacionais potencialmente diferentes, assim como *middleware*, protocolos de comunicação, modelos de dados, tipos de aplicativos e políticas de segurança.

Nessa situação, em função da mobilidade ou de determinações administrativas, diferentes instâncias de IoT comumente se sobrepõem no espaço físico ou na lógica de endereçamento da Internet, ocasiões em que surge a oportunidade de interoperação entre as diferentes instâncias de IoT, o que também responde a requisitos e necessidades funcionais e de desempenho envolvendo tais instâncias.

Consideradas tais necessidades, a proposta desta tese é concebida na forma de um modelo de interoperação com dois componentes principais, um deles devendo ser agregado na forma de biblioteca de funções a cada *middleware* controlador de instância de IoT, enquanto o outro é o serviço P2P de identificação, roteamento e busca (*lookup*) pelo qual se passam a comunicações em situação de interoperação entre duas ou mais instâncias de IoT.

Tais elementos são descritos neste capítulo, começando com uma discussão das necessidades funcionais e de desempenho, seguida da proposta propriamente dita do Modelo P2PIoT com seus detalhes estruturais, componentes e sequências de funcionamento, assim como a representação de abstrações utilizadas e a discussão de outros fatores ligados à mobilidade e ao tempo.

3.1 Caracterização de Requisitos Funcionais e de Desempenho

Para caracterizar melhor os requisitos e necessidades, cabe descrever situações comuns que envolvem mais de uma instância de IoT operando simultaneamente e paralelamente. A seguir, são descritas três situações simples, mas ilustrativas e generalizáveis, que permitem apresentar algumas questões motivadoras da proposta apresentada nesta tese, assim como descrever comparativamente em termos gerais a nossa proposta com relação às possibilidades existentes baseadas em outros modelos.

3.1.1 Análise de Situações de Interoperação de IoT

Um exemplo simples e muito comum é a situação em que determinado dispositivo deve ser compartilhado entre duas aplicações diferentes, cada uma delas encontrando-se sob um *middleware* diferente, conforme exemplo da Figura 3.1.

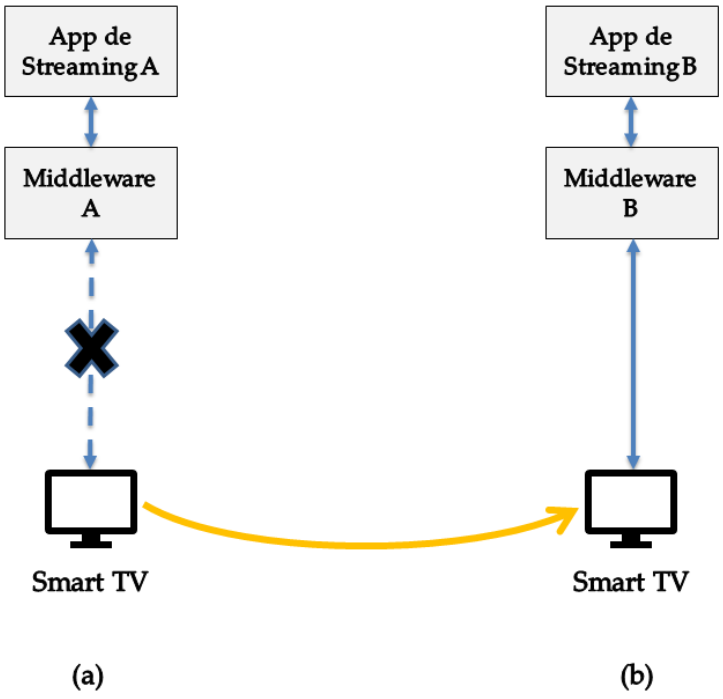


Figura 3.1: Dispositivo deslocado de um *middleware* IoT para outro

Nessa Figura 3.1, apresenta-se a corriqueira situação de uma TV conectada que é usada para *streaming* de cinema em uma casa. Tal TV ora está sendo usada por um aplicativo de *streaming* de cinema sob controle do sistema operacional, com ou sem *middleware* específico de IoT, do aparelho móvel de determinado usuário, ora passa ao controle do aparelho móvel de outro usuário para ser usada pelo aplicativo de *streaming* deste último.

Nesse caso, a solução simples para o compartilhamento do dispositivo consiste em desconectá-lo do primeiro *middleware* e reconectá-lo ao outro *middleware*, o que no sentido estrito nem chega a ser uma função de interoperação entre os dois diferentes *middlewares*, posto que se trata de operação com intervenção manual externa aos dois *middlewares* IoT.

Outro exemplo é aquele em que, diante da mesma necessidade, existe um *middleware* que tem em suas bibliotecas de funções, um módulo para controlar outro *middleware*. Assim, conforme a Figura 3.2, configura-se outra solução clássica para as questões de interoperação, consistindo em um determinado *middleware* centralizar o controle da interoperação, o que é caracterizado por conhecidos problemas de complexidade e inchaço do citado *middleware*, dificuldade de desenvolvimento e evolução dos componentes, dependência do componente central, etc.

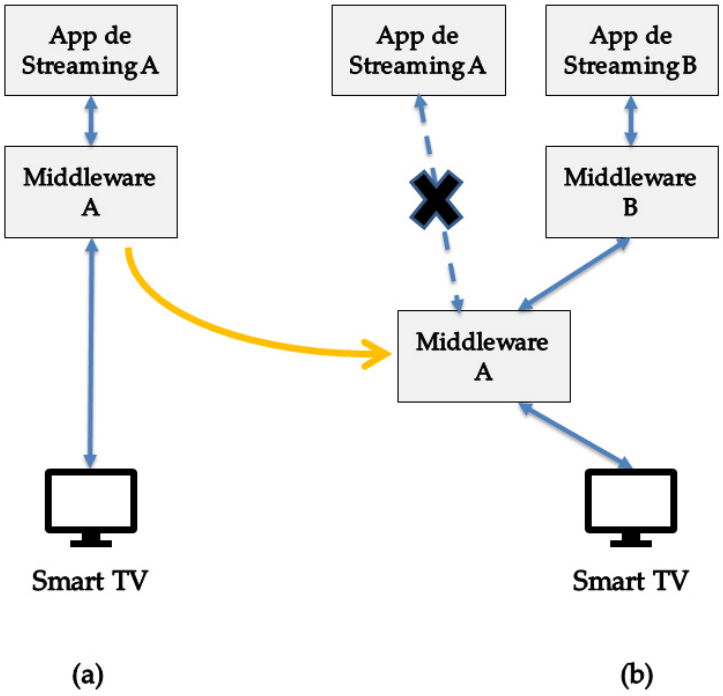


Figura 3.2: Um *middleware* IoT controla outro *middleware* para acessar o dispositivo

Imagine-se, por exemplo, a necessidade permanente de acrescentar novos módulos controladores ao *middleware*, o versionamento dos módulos, a concorrência com outros *middlewares* pela posição de centralidade, a multiplicação dos esforços, dentre outras questões conhecidas do domínio de desenvolvimento e operação de software. Além disso, aponta-se nas soluções centralizadas a vulnerabilidade de serem convenientes para ataques de interceptação que permitem ao atacante manipular diretamente as informações de todo o sistema atacado [69].

Em contraposição, conforme ilustra a Figura 3.3, esta tese propõe um modelo para a interoperação de diferentes instâncias de IoT usando serviços entre entidades pares (P2P, da sigla em Inglês para a expressão *peer-to-peer*) e fornecendo operações para federar essas instâncias em cenários fixos e móveis.

Nesse caso, cada *middleware* tem igual poder e cada um integra tão somente um conjunto de módulos de software para interoperar com os demais *middlewares* que estejam dotados dos recursos P2P de maneira similar.

Assim, o compartilhamento do dispositivo passa pela interação entre os *middlewares* envolvidos, mas de uma maneira relativamente transparente para a aplicação de *streaming* que usa os serviços resultantes da interoperação entre as instâncias de IoT.

Em contraste com a solução centralizada, não há um pressuposto de confiabilidade quanto aos participantes de uma rede P2P, posto que nesta os meios de robustez são incorporados aos protocolos de sobreposição e as informações relacionadas aos participantes são distribuídas entre os pares de maneira descentralizada, sem a necessidade de um servidor dedicado que possa ser atacado por interceptação [69].

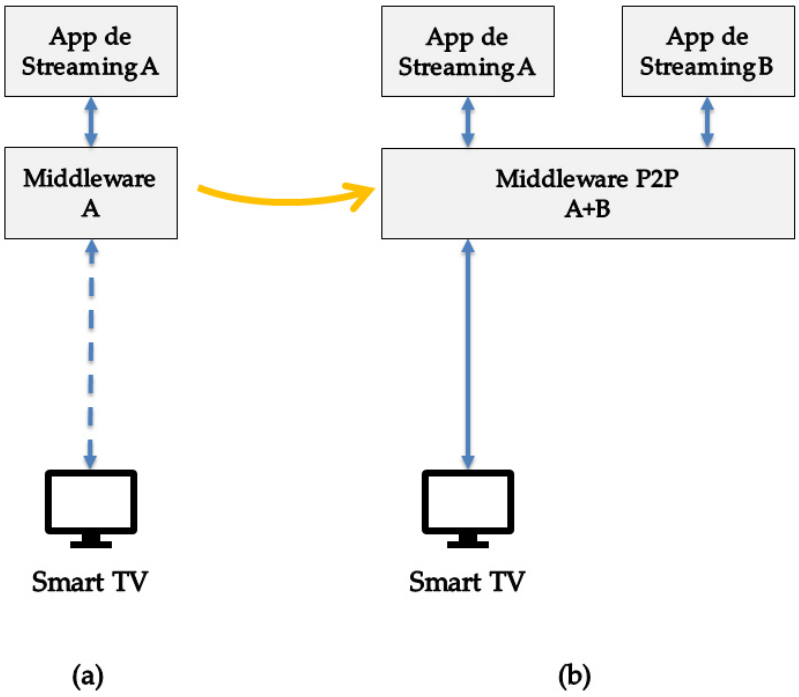


Figura 3.3: Par de *middlewares* IoT Coordena o Acesso ao Dispositivo

3.1.2 Especificação de Requisitos Funcionais e de Desempenho

Para responder às necessidades discutidas nessa análise de situações, ou seja, para obter como efeito emergente a transparência nas interações entre instâncias de IoT, bem como contribuir para a escalabilidade na federação dessas instâncias de IoT, a arquitetura proposta nesta tese suporta serviços para mesclar e dividir instâncias de IoT, bem como para mover, replicar, restaurar e excluir dados, entidades de software e abstrações de dispositivos nessas instâncias de IoT. Então, deseja-se atender a necessidades funcionais assim especificadas:

- Mesclar duas ou mais instâncias;
- Mover dados e dispositivos de uma instância para outra;
- Copiar e fundir dados a partir de instâncias originalmente diferentes e separadas;
- Copiar dados de dispositivos visando substituir ou duplicar tais dispositivos, com a restauração dos dados nos novos dispositivos inseridos;
- Dividir uma instância existente, separando os componentes para duas ou mais instâncias.

Outro ponto de interesse é a questão do desempenho, haja vista que se requer da proposta P2P desta tese apresentar funcionalidade e desempenho similares a soluções centralizadas, o que, por um lado, espera-se que resulte da maior simplicidade do software P2P que é requerido em cada *middleware* IoT, mas cuja adaptação e versionamento é relativamente mais simples do que produzir adaptadores para sistemas centralizados, conforme discutido mais acima. Por outro lado, funcionalidade e desempenho dependerão, no caso P2P, dos fluxos de informação e dos tempos de execução das operações P2P, assuntos que são discutidos como parte da proposta desta tese a seguir.

3.2 Modelo Proposto P2PIoT

Reiterando a discussão inicial, no modelo proposto nesta tese a ideia central é a de federar diferentes instâncias de IoT em torno de um arcabouço P2P.

Para tanto, considera-se uma estrutura genérica da instância de IoT, conforme mostra a Figura 3.4, em cujo *middleware* o Modelo P2PIoT proposto nesta tese integra tão somente um conjunto de módulos de software para interoperar com os demais *middlewares* dotados de maneira similar.

Assim, todos os *middlewares* controladores em cada instância de IoT são considerados como tendo igual poder e cada um proverá serviços de interoperação comuns aos demais.

Na realidade, apenas para facilitar a inicialização do universo P2P, pelo menos umas das instâncias deverá ter um papel de abrigar a semente (*seed*) do arcabouço P2P, mas tal papel pode ser atribuído igualmente a qualquer instância IoT, embora tipicamente seja atribuído a uma instância sobre a qual um poder administrativo possa garantir a permanência dos serviços necessários à inicialização e o ponto de referência (URL permanente) para as demais entidades.

Partindo desses pressupostos, são descritos os módulos componentes estruturais do modelo e são apresentados os diagramas de descrição do funcionamento dos serviços providos por tais módulos.

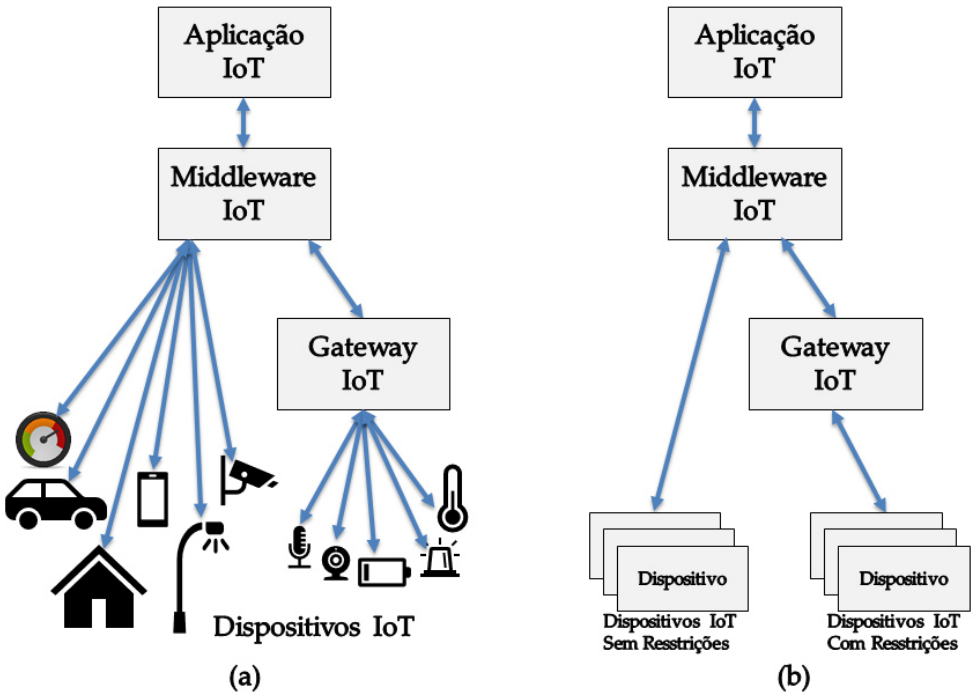


Figura 3.4: Estrutura Genérica da Instância de IoT (a), com Representação Modular em (b).

3.2.1 Módulos Componentes da Estrutura do Modelo P2PIoT

Em consideração aos requisitos de interoperação vislumbrados. bem como aos pressupostos de transparência, escalabilidade e desempenho sintetizados acima, foi concebido o Modelo P2PIoT desta tese, conforme mostra a Figura 3.5, incluindo em sua composição, para a interoperação de diferentes instâncias de IoT, os seguintes módulos:

- **Módulo de Instância Par**: trata-se do próprio módulo de *middleware* que controla cada instância de IoT. A estrutura deste módulo não é objeto desta tese, sendo apenas considerado aqui pelo fato de que precisa ser completado para receber os módulos propostos nesta tese que são designados em seguida nesta lista;

- **Módulo de Serviços de Interoperação P2P**: é o módulo que contém os códigos de programas dos serviços de interoperação (*merge*, *split*, *copy*, *move*, *delete*). Este módulo deve ser acoplado diretamente ao *middleware* interno da instância de IoT, ou seja, ao **Módulo de Instância Par**. Tal acoplamento deve ser forte, haja vista que o **Módulo de Serviços de Interoperação** é obrigado a utilizar a representação abstrata dos recursos do **Módulo de Instância Par** e fazer a conversão das funcionalidades de interoperação P2P em operações locais do *middleware* da instância de IoT participante. Por outro lado, os serviços de interoperação providos utilizarão as operações do **Módulo de Serviços de Busca P2P** para as interações com as outras instâncias pares federadas;

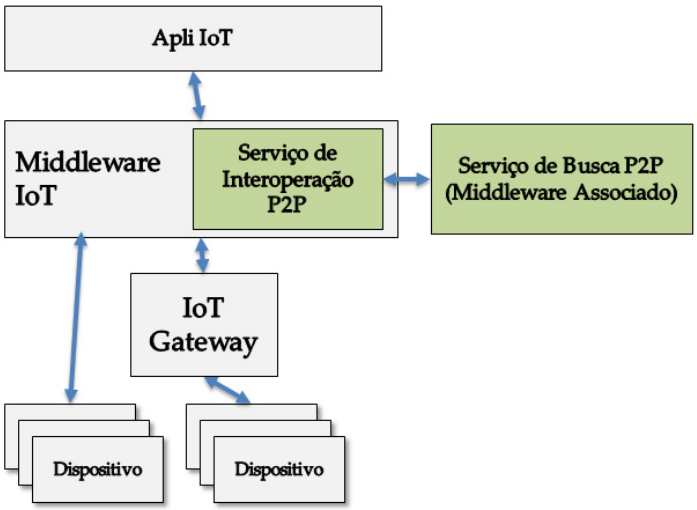


Figura 3.5: Módulos do Modelo P2PIoT

- **Módulo de Serviços de Busca P2P**: este módulo constitui o serviço de busca P2P (*lookup*) sobre qual se sobrepõem serviços de interoperação de IoT. No modelo proposto nesta tese, este módulo pode ser fracamente acoplado ao **Módulo de Serviços de Interoperação P2P**, devendo prover um sistema de identificação P2P, bem como serviços de inserção e retirada de nodos e recursos P2P, serviços de manutenção da integridade P2P e, obviamente, o serviço de busca de recursos P2P.

Assim, no que se relaciona ao **Módulo de Instância Par**, o pressuposto é a existência de *middlewares* de IoT que possam ser adaptados para receber os módulos de interoperação aqui propostos. Não são impostos requisitos outros cada *middleware* possivelmente participante, a não ser que tal *middleware* possua características que são comuns aos diversos já existentes na atualidade, ou seja, basicamente uma representação abstrata de recursos e serviços internos que suportem o registro das entidades (dispositivos e aplicações) e os fluxos de informação, comando e controle entre as aplicações e os dispositivos de IoT, ou seja, meios de interoperação internos da instância de IoT e particulares do seu *middleware* e provendo os fluxos de informação, identificações necessárias e abstrações de software para que a operações ocorram de forma correta.

3.2.2 Módulos Opcionais de Suporte ao Modelo P2PIoT

Vale notar que, em função de necessidades de escalabilidade ou de desempenho, muitas vezes o *middleware* da instância de IoT necessita de outros módulos que são separados quanto à lógica da funcionalidade provida. Em alguns casos, existem módulos que são totalmente integrados no modulo principal do *middleware*, mas outras tantas vezes são módulos a parte que funcionam em interoperação direta com o *middleware*, como módulos de gerenciamento, controle de acesso, de segurança, dentre outros.

Um possível módulo desse tipo, conforme proposto no trabalho [36], é aquele que constitui um serviço de dados distribuído para responder a características de *bigdata* de IoT, ou seja, requisitos de grande velocidade de geração e de fluxo de dados, bem como grande volume e variabilidade destes dados. Esse mesmo tipo de componente pode ser usado para armazenar informações referentes ao modelo de interoperação proposto nesta tese.

Outro módulo de *middleware* dessa classe é o chamado *gateway* IoT, empregado quando se precisa de um módulo compacto do *middleware* IoT para ficar mais perto dos dispositivos, nos casos em que há concentração de dispositivos em determinada área, com possivelmente uma grande quantidade de protocolos que perturbaria o funcionamento correto de um *middleware* de controle principal da instância de IoT. Em outras situações, tal módulo visa responder às necessidades de segmentação e modularização nessas redes. É interessante observar que o *gateway* IoT, conforme descrito em [18], introduz a ideia de computação na borda (*edge*) para racionalizar a organização da instância de IoT, provendo maior escalabilidade e melhor desempenho. Sendo o *gateway* um módulo compacto do *middleware* IoT, seria possível prover no próprio *gateway* os serviços de interoperação propostos nesta tese. Entretanto, como cada *gateway* tem cobertura de ação limitada a uma parte da instância de IoT, parece ser mais interessante que a interoperação seja provida pelo *middleware* principal integrador dos vários *gateways* da instância de IoT.

Já no que se refere ao **Módulo de Serviços de Interoperação P2P**, os requisitos de serviços são comumente encontrados em sistemas existentes, em especial no sistema Chord [4] que é um dos principais paradigmas, senão o principal, nesse campo do conhecimento. Em tese, qualquer sistema que dê suporte aos mesmos serviços de registro, manutenção e busca em P2P, poderia ser utilizado, inclusive algum sistema P2P com mecanismo de consenso como *blockchain*, valendo entretanto observar que as características do consenso e registro distribuído do *blockchain* ainda são consideradas objeto de investigação quanto a sua aplicação em IoT [8]. Por essa razão, e considerada a situação do Chord no domínio de conhecimento, o modelo de interoperação proposto nesta tese obedece em linhas gerais à conceituação de elementos do sistema Chord, que inclusive é utilizado no capítulo 4 desta tese como parte do protótipo utilizado na validação da proposição da tese.

3.2.3 Estrutura Federativa Emergente do Modelo P2PIoT

Definida a modularização do modelo P2PIoT, com seus três componentes, **Módulo de Instância Par**, **Módulo de Serviços de Interoperação P2P** e **Módulo de Serviços de Busca P2P**, constitui-se cada instância de IoT em conformidade ao modelo da Figura 3.5, podendo assim cada um ser membro, ou par, na federação a ser possivelmente gerada segundo o modelo P2PIoT.

Estando cada uma das instâncias IoT pares dotadas dos módulos apresentados na Figura 3.5, um conjunto de instâncias de IoT pode ser agregado em torno do serviço de busca P2P utilizado via **Módulo de Serviços de Busca P2P**, conforme mostra a Figura 3.6. Desse modo, com os serviços de busca e interoperação providos, as operações federativas podem ser realizadas para instâncias heterogêneas de IoT em cenários fixos e móveis.

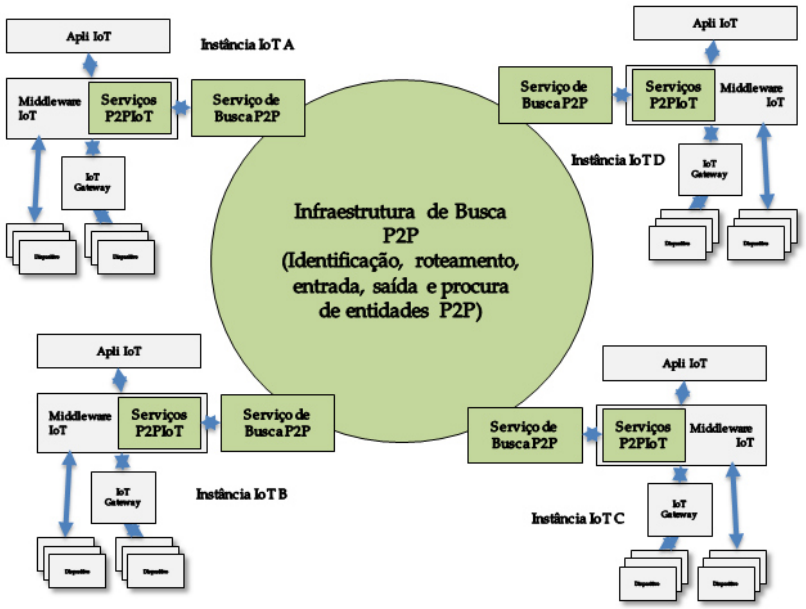


Figura 3.6: Modelo P2PIoT mostrando instâncias heterogêneas de IoT federadas

Os serviços de interoperação são providos então com cobertura sobre todas as instâncias de IoT federadas que, de fato, na perspectiva do serviço de busca (*lookup*) P2P, constituem um ambiente unificado do ponto de vista da identificação, roteamento e procura por entidades participantes. Neste ambiente, as funcionalidades de interoperação ocorrem entre todos os federados, de modo a permitir mesclar e dividir instâncias de IoT, bem como mover, replicar e excluir dados da IoT, ou de suas entidades de software e abstrações de dispositivos.

3.2.4 Serviços Federativos Emergentes do Modelo P2PIoT

Assim, o modelo é organizado e dotado de funcionalidades que permitem as seguintes ações no conjunto de instâncias de IoT federadas em determinado momento, seja a federação motivada pela mobilidade, seja por determinação administrativa:

- Mapear os identificadores de entidades e recursos locais a cada instância de IoT em identificadores válidos no serviço de busca P2P, assim como dissociar tais identificadores (*merge*, *split*);
- Agrupar (*merge*) e desagrupar (*split*) instâncias de IoT;
- Substituir dispositivos com recuperação da base informacional completa de cada um (*move*);
- Copiar a base informacional de um dispositivo para outro (*copy*);
- Consolidar bases informacionais de vários dispositivos, para agregação de conhecimento (*copy*);
- Duplicar dispositivos de uma instância para outra (*move*);
- Transportar dispositivo de uma instância de IoT para outra (*move*);
- Fazer um dispositivo participar em duas instâncias diferentes de IoT, com entrada e saída dinâmica (*merge*).

3.2.5 Artefatos de Software do Modelo P2PIoT

Em termos dos artefatos de software, o modelo proposto P2PIoT requer então tornar cada *middleware* IoT capaz de gerenciar seus recursos, não apenas com seu próprio subsistema de identificação de entidades e objetos, mas também com os identificadores usados no serviço de busca P2P.

Para tanto, um agente de software deve ser adaptado dentro do *middleware* para mapear o endereçamento P2P. Assim, é possível implementar o modelo sobreposto a um serviço P2P já consolidado e conhecido, como por exemplo o Chord [4]. Tal solução é inclusive adotada no processo de validação do P2PIoT, conforme relatado no capítulo 4, haja vista que o Chord é um software P2P que apresenta as funcionalidades requeridas, estando disponível para ser utilizado na pesquisa.

Nessas condições, os serviços de interoperação do P2PIoT (*merge*, *split*, *copy*, *move*, *delete*) podem ser realizados na forma de uma biblioteca de funções virtuais em um artefato de software fortemente acoplado ao *middleware* de IoT.

3.3 Diagramas de Sequência dos Serviços de Interoperação do P2PIoT

Haja vista que a concretização do modelo proposto P2PIoT é matéria de interação entre módulos de software inseridos no *middleware* de cada instância de IoT que poderá vir a se federar com outras instância armadas com o mesmo aparato de software.

Por essa razão, é imprescindível especificar as interações entre tais módulos de software, o que é o objeto desta seção da tese, na forma de diagramas de sequência comentados. Em tal tipo de diagrama, os retângulos superiores representam cada entidade participante, sendo cada uma identificada pelo nome no interior do retângulo. As linhas tracejadas partindo dos retângulos e seguindo na vertical para baixo representam o desenrolar dos acontecimentos no tempo local percebido pela entidade. Entre tais acontecimentos, há a chegada de eventos e mensagens representada pelas flechas chegando na linha vertical, ações internas da entidade representadas por retângulos sobrepostos às linhas tracejadas, e ações externas e mensagens enviadas representados por linhas com flechas saindo da linha vertical.

3.3.1 Diagrama Geral de Operações

Na Figura 3.7 é apresentado o diagrama geral de sequência das operações realizadas no P2PIoT. Para efeito de simplificação do diagrama, a linha do tempo à esquerda representa em uma única entidade P2PIoT a entidade de software que está presente em qualquer instância de IoT. Como essa entidade é comum a todas as instâncias, ela foi fatorada e representada por uma única linha vertical do diagrama. Já as entidade denominadas NÓ1 e NÓ2 representam duas diferentes instâncias de IoT.

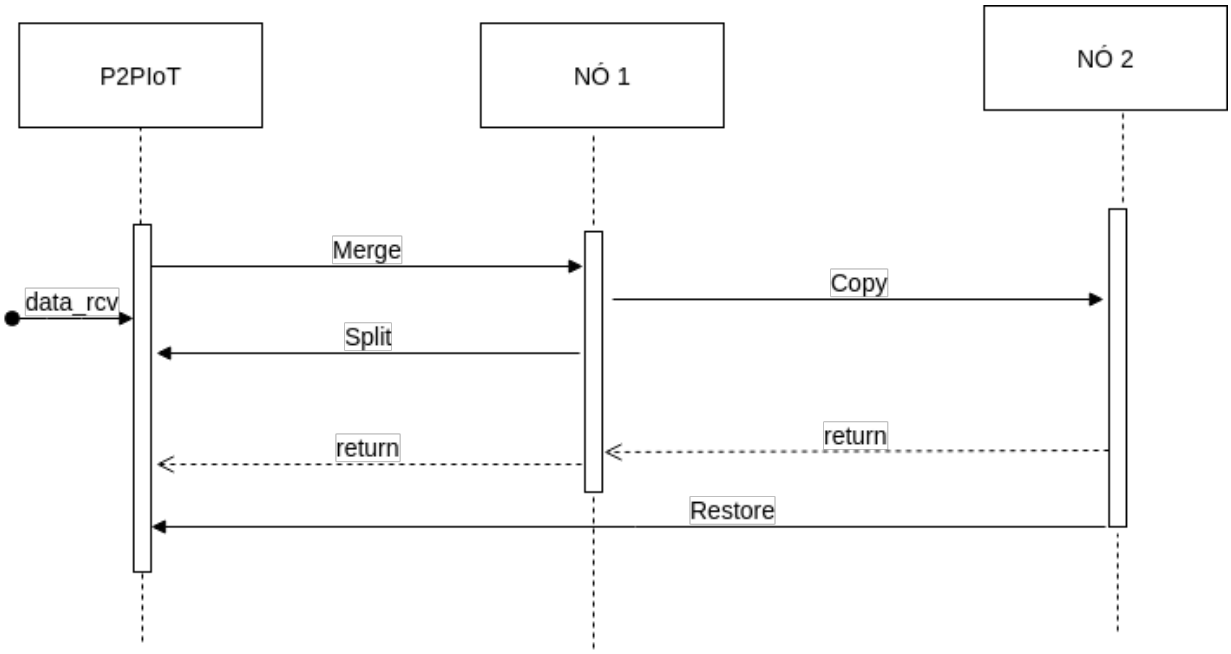


Figura 3.7: Fluxo das operações desenvolvidas do P2PIoT

A primeira interação ocorre então por iniciativa do Módulo P2PIoT presente em qualquer das instâncias, ou seja, ou por receber um comando administrativo ou por detectar automaticamente a presença da outra instância, tal Módulo inicia a mesclagem acionando a função *merge* e respectiva mensagem comandando o NÓ1 a realizar a junção P2P com o NÓ2.

Feita a junção, é possível copiar, função *copy*, elementos de uma instância para outra, apagar elementos, função *delete*, bem como restaurar para a instância de origem um elemento que tenha sido copiado para outra instância e depois apagado na instância de origem.

A última interação apresentada é o comando, ou ação automática resultante de detecção de afastamento da outra instância, para que o NÓ1 se desvincule do arcabouço P2P federado e passe a operar sozinho, o que corresponde à função *split*.

Embora representada ao final do diagrama, a separação, ou mais precisamente a saída da federação, pode ocorrer a qualquer momento na linha de tempo, por iniciativa do Módulo P2PIoT de qualquer das instâncias.

As operações discutidas acima requerem a execução de algoritmos básicos do módulo P2P do modelo proposto, o que justifica desde já especificá-los.

O Algoritmo 1 é apresentado na forma de pseudocódigo para identificação de nós sucessores e predecessores, pressuponde a utilização de um Módulo P2P que adote a terminologia do sistema Chord, adotando-se especificamente os seguintes termos: sucessor é o próximo nó presente no anel P2P e predecessor é o nó anterior ao que será inserido na federação de instâncias de IoT.

Algorithm 1 Pseudocódigo para encontrar os identificadores id dos nodos sucessor e predecessor para inserção de novo nodo na federação

```
1:  procedure FIND_SUCESSOR(id)              ▷ Solicita ao nó n encontrar o id do sucessor
2:      n' ← find_precessor(id);
3:      return n'.sucessor
4:
5:  procedure FIND_PREDECESSOR(id)           ▷ Solicita ao nó n encontrar o id do predecessor
6:      n' ← n;
7:      while id ∉ (n', n'.sucessor) do
8:          n' ← n'.closest_position(id);
9:          return n'
10:
11: procedure CLOSEST_POSITION(id)           ▷ Retornar o nó mais próximo ao id informado
12:     for i = m downto 1 do
13:         if position[i].node ∈ (n,id) then
14:             return position[i].node;
15:     return n;
```

Algorithm 2 Pseudocódigo para operações de *backup* e de nodos pressupondo o suporte do sistema P2P Chord

```
1:  procedure JOIN(n′)                          ▷ Função que insere um novo nó ao anel CHORD
2:      predecessor = NULL;
3:      sucessor = n′.find_sucessor(n);
4:
5:  procedure STABILIZE                  ▷ Pergunta ao nó n para encontrar o id do predecessor
6:      x = sucessor.predecessor;
7:      if x ∈ (n, sucessor) then
8:      sucessor.notify(n);
9:
10: procedure NOTIFY(n′)                                ▷ Função para procurar o predecessor
11:     if predecessorisNULLorn′ ∈ (predecessor, n) then
12:         predecessor = n′;
13:
14: procedure STORE_NODE_ID(n)                ▷ Salva o id de cada nó inserido na federação
15:     next = next + 1
16:     if next > m then
17:         next = 1;
18:     table[next] = find_sucessor(n + 2^(next−1));
```

3.3.2 Diagrama da Operação de Mesclagem

Já a Figura 3.8 apresenta detalhes da mesclagem ou federação, com os comando de junção *join* preparatórios sendo enviados às instâncias NÓ1 e NÓ2, seguidos do comando merge propriamente dito. Para melhor entendimento, a entidade federada pela junção de NÓ1 e NÓ2 é representada pelo NÓ3, haja vista que as instâncias originais não deixam de existir, mas tão somente passam a interoperar na entidade federada emergente da mesclagem.

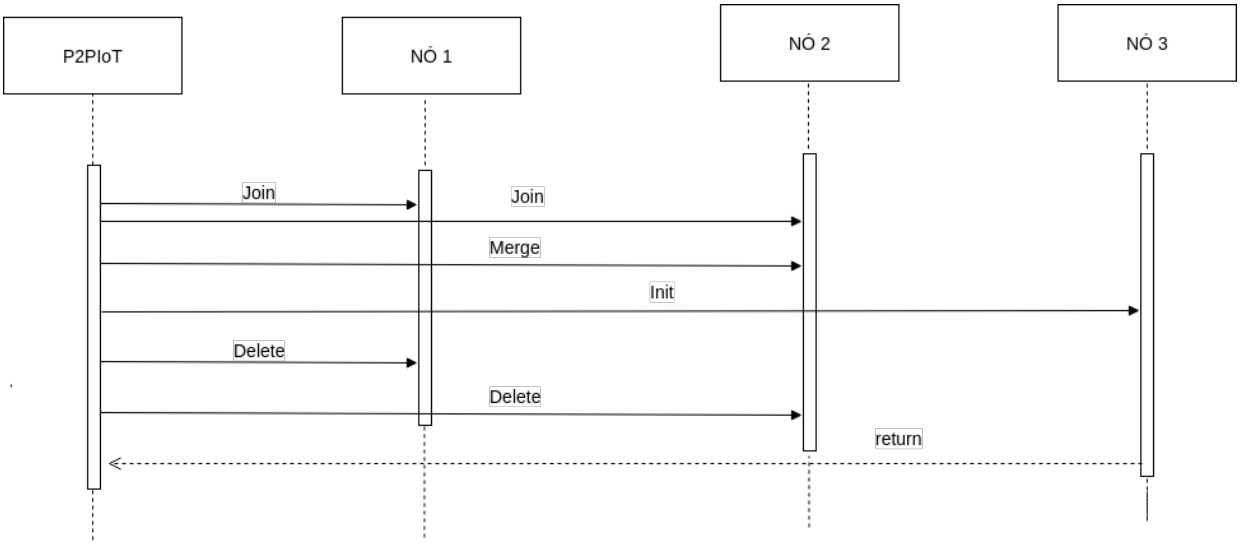


Figura 3.8: Fluxo das operação de Merge do P2PIoT

3.3.3 Diagrama da Operação de Separação

Continuando com a mesma representação da entidade federada NÓ3, a Figura 3.9 mostra detalhes do encerramento da federação, com a separação das instâncias, ou seja na verdade a saída da federação. O processo inicia com o comando *split* ao NÓ3 federado, seguido de reinicialização dos nodos NÓ1 e NÓ2, resposta destes explícitas à entidade P2PIoT.

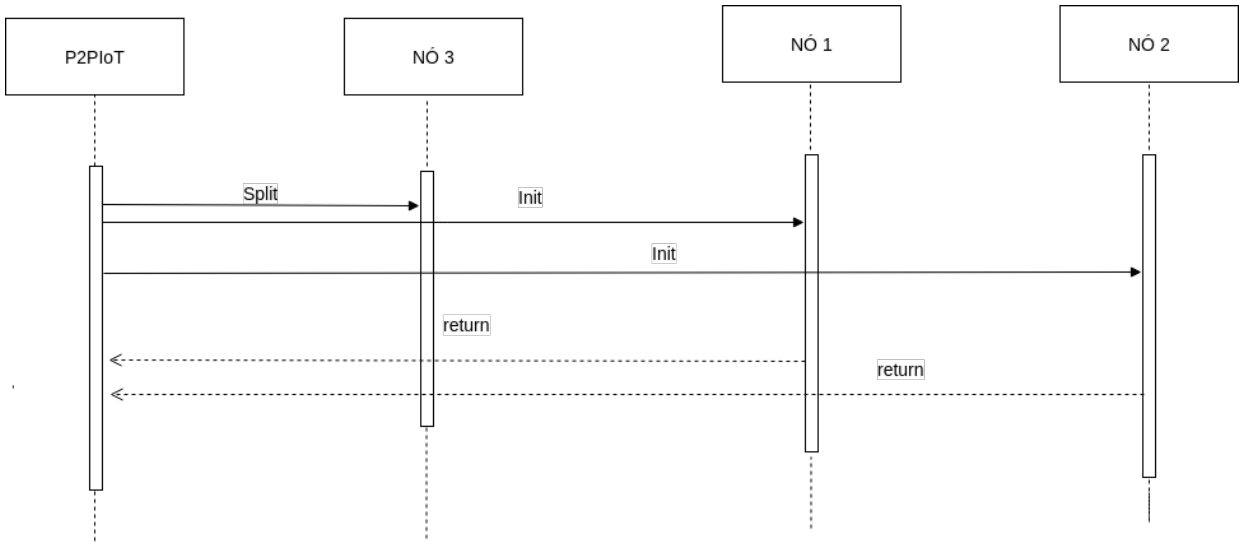


Figura 3.9: Fluxo das operação de Split do P2PIoT

3.3.4 Discussão das demais Operações

Não são destacadas na forma de diagramas as demais operações pois tratam-se mais de questões de manipulações de estruturas de dados representativas de dispositivos e de aplicações nas diversas instâncias federadas. vale notar que, dada a heterogeneidade das instâncias, é necessário que na adaptação do Módulo P2PIoT a cada instância seja também realizada toda a conversão de comandos de interoperação definidos no contexto P2P, bem como os respectivos parâmetros desses comandos, para comandos e abstrações de dados que façam sentido e estejam operantes no contexto específico, ou seja, do *middleware* de cada uma das instâncias adaptadas ao Modelo P2PIoT proposto nesta tese.

A descrição das abstrações de capacidades esperadas pelo Modelo P2PIoT é o assunto das seções seguintes deste capítulo.

3.4 Abstrações de Capacidades dos Dispositivos IoT

Para que o modelo de interoperação proposto funcione adequadamente em instâncias de IoT heterogêneas, seus serviços devem ser projetados para operar com uma representação abstrata comum das entidades que podem ser monitoradas e controladas nas diferentes instâncias de IoT.

Embora esta tese use uma representação existente que foi desenvolvida em trabalhos anteriores correlacionados, esta representação é descrita aqui para tornar completa a descrição da nossa proposta, uma vez que esta representação de entidade abstrata deve estar disponível em cada módulo contendo os serviços de interoperação em nossa proposta. Naturalmente, a representação abstrata requer sua retificação, ou seja, ela deve ser adaptada para cada *middleware* específico que gerencia alguma instância da IoT, o que significa codificação de software necessária para adaptar nosso projeto ao caso concreto de uma instância da IoT.

Esta seção compreende inicialmente uma breve visão geral das necessidades de abstração de capacidades, um resumo da especificação UPnP e dos serviços REST / SOAP que são bases da representação definida e, em seguida, a apresentação propriamente dita da API para a representação abstrata comum das entidades IoT necessárias para o modelo de interoperação proposto nesta tese.

3.4.1 Necessidades Gerais de Abstrações de Capacidades

Essa representação abstrata usada na proposta desta tese é descrita em [61], na forma de uma interface de programação de aplicativos (API) para acessar serviços dentro da internet das coisas (IoT) através dos protocolos REST e SOAP. Essa API fornece métodos e procedimentos para permitir seu uso para executar operações de controle e de monitoramento de eventos da IoT que envolvam dispositivos heterogêneos.

A fim de abranger a diversidade de dispositivos, a API proposta introduz um modelo de abstração uniforme que constitui uma visão padrão comum para gerenciar objetos. Uma interface abstrata de serviços do dispositivo é então disponibilizada em vez dos comandos do dispositivo, fornecendo assim acesso transparente aos recursos dos dispositivos e ocultando os aspectos físicos dos dispositivos provedores concretos.

A API foi projetada para facilitar o gerenciamento remoto de objetos inteligentes de IoT e foi implementada como um módulo do *middleware* UIoT existente, fato que facilita a validação de nosso design, pois essa tese usa para fins de validação, um protótipo baseado nesse mesmo *middleware*, como descrito no capítulo 4.

Outro aspecto interessante da representação abstrata e dos respectivos fluxos de programas é o tempo necessário para executar seus serviços, um tema que influencia atividades de interoperação durante a fusão de instâncias de IoT e depois na IoT federada, especialmente se a mobilidade for considerada e a geolocalização for usada por instâncias IoT como meio de detectar outras instâncias de IoT a serem mescladas. A influência do tempo e da mobilidade em nossa proposta é considerada com mais detalhes na seção 3.5. Por enquanto, é interessante ressaltar que a avaliação experimental em [61] mostra que a representação abstrata de entidades como serviços REST oferece tempo de resposta mais rápido e consome menos recursos do que serviços SOAP semelhantes.

Considerando o número potencial de objetos da IoT e sua heterogeneidade, é importante

desenvolver e implantar interfaces de programação de aplicativos da IoT (IoT API) para permitir acesso direto a recursos e operações de dispositivos. Sem essas APIs, a magnitude e a longevidade da IoT podem ficar comprometidas.

A API IoT proposta é projetada de acordo com o padrão Universal Plug and Play (UPnP) [70] e padrões de acesso a serviços da Web, permitindo seu desenvolvimento por meio da arquitetura REpresentational State Transfer (REST) e do Simple Object Access Protocol (SOAP). A API proposta considera os cenários de uso de IoT apresentados em trabalhos anteriores [71] para especificar os serviços necessários que são projetados e desenvolvidos para fornecer acesso uniforme a recursos e operações, apesar da diversidade de dispositivos de IoT. Além disso, o design da API é orientado para otimizar a utilização de recursos computacionais para permitir o controle dinâmico de dispositivos da IoT por meio de um sistema de consulta simples.

A definição geral de um serviço de IoT vem de [72], que afirma: "Um serviço de IoT é uma transação entre duas partes, o provedor de serviços e o consumidor de serviço. Isso faz com que uma função prescrita permita a interação com o mundo físico medindo o estado das entidades ou iniciando ações que causarão uma mudança nas entidades ".

Como mostrado em [73], a identificação única de objetos, a representação e o armazenamento de informações trocadas é a questão mais desafiadora para os *middlewares* de IoT. Uma das possíveis soluções para superar esses desafios é criar uma camada de serviço no *middleware* que forneça informações históricas sobre dispositivos e ações em um ambiente de IoT. No entanto, a maioria das pesquisas descreve a prestação de serviços em IoT com base em conceitos gerais ou nas necessidades de uma aplicação específica, e não através de um padrão bem estabelecido. Existem várias propostas para o tipo e a lista específica de serviços a serem fornecidos. As referências [74] [75] definem os seguintes elementos de serviço: micro-pagamento, armazenamento e recuperação, serviço de apresentação, serviço de sessão, serviço de transporte, relatório de evento, presença de objetos conectados, localização e status, mobilidade, controle de QoS, transmissão de dados, associação de segurança criação e portador IP básico. Autores em [26] apresentam um *middleware* baseado em SOA com dois componentes de serviço: composição de serviços e gerenciamento de serviços. Em [76], uma arquitetura orientada a serviços é proposta para IoT, incluindo quatro camadas denominadas: sensoriamento, rede, serviço e interface. A camada de serviço inclui componentes para descoberta de serviços, composição de serviços, gerenciamento de confiabilidade e suporte para as interações entre serviços. Um modelo de ontologia semântica é proposto em [77] para representar dispositivos, recursos e serviços, com componentes que devem estar presentes em um modelo de domínio de serviço IoT, muito semelhante à definição UPnP.

Este breve levantamento mostra a necessidade de uma definição genérica e padronizada para acessar serviços de gerenciamento de IoT e estar em conformidade com os padrões e tecnologias estabelecidos. Assim, a API considerada nesta tese procura fornecer informações sobre dispositivos e ações em uma rede de IoT. A API é baseada no modelo de entidade UPnP, bem como nas abordagens REST e SOAP para comunicações e nos formatos de dados padrão JavaScript Object Notation (JSON) e Extensible Markup Language (XML).

3.4.2 Universal Plug and Play – UPnP

Universal Plug and Play [70] é um conjunto de protocolos de rede que permite que dispositivos em rede estabeleçam automaticamente configurações de trabalho com outros dispositivos, incluindo a descoberta da presença uns dos outros na rede e a configuração de serviços que cada dispositivo oferece para interoperar com outros dispositivos e aplicativos.

De fato, o UPnP é uma pilha de protocolo baseada em TCP/IP projetada como um padrão para acesso transparente universal a recursos de objetos lógicos cujas operações compreendem 6 fases principais, denominadas Endereçamento, Descoberta, Descrição, Controle, Monitoração de Eventos e Apresentação. Além disso, o UPnP propõe uma arquitetura de dispositivos precisa para a abstração de dispositivos usados para executar essas operações em todo o SSDP, SOAP e GENA.

3.4.3 REpresentational State Transfer – REST

REST é definido em [19] como um conjunto coordenado de restrições arquiteturais que tentam minimizar a latência e a comunicação de rede, enquanto maximizam a independência e a escalabilidade das implementações de componentes.

A abstração de chave no REST é o recurso, que é um mapeamento conceitual para um conjunto de entidades. Qualquer coisa que possa ser nomeada pode ser um recurso e a semântica associada a um recurso deve ser estática, embora as entidades por trás de um recurso possam mudar com o tempo [78].

Conforme descrito em [78], o REST inclui três tipos de elementos arquiteturais: tipos de dados, elementos de conexão e elementos de processamento. No REST, os componentes de arquitetura trocam informações transferindo representações do estado atual ou desejado dos elementos de dados. Essas transferências usam os quatro tipos de mensagem HTTP existentes e acessam os serviços disponíveis fazendo referência aos URLs correspondentes vinculados a esses serviços. Assim, o REST fornece uma interface simples e direta para acessar serviços da web. As respostas desses serviços podem ser apresentadas em vários formatos, como CSV (Command Separated Value), JSON (JavaScript Object Notation) e RSS (Really Simple Syndication), portanto, a saída está pronta para ser analisada diretamente no aplicativo consumidor de serviço.

3.4.4 Simple Object Access Protocol – SOAP

O SOAP é um protocolo baseado em XML para mensagens e chamadas de procedimento remoto (RPCs). Em vez de definir um novo protocolo de transporte, o SOAP funciona em protocolos de transferência existentes, como HTTP, SMTP e MQSeries. A especificação SOAP define um modelo que determina como os destinatários devem processar as mensagens SOAP. O modelo de mensagem também inclui os atores, que indicam quem deve processar a mensagem. Uma

mensagem pode identificar atores que indicam uma série de intermediários que processam as partes da mensagem para eles e repassam o restante [79].

Como meio de lidar com possíveis complexidades no XML usado para fazer solicitações e receber respostas, no SOAP, o WSDL (Web Services Description Language) é usado para fornecer uma definição de como o serviço da Web funciona, reduzindo assim o esforço necessário para criar, solicitar e analisar a resposta.

3.4.5 Componentes da API REST/SOAP para serviços IoT

A API REST/SOAP para serviços IoT oferece uma abstração lógica de dispositivos, a base para todos os serviços fornecidos. O modelo abstrato da API é organizado com camadas, módulos e componentes. Os fluxos de informações em todos esses elementos constituem rotas REST e SOAP para descoberta de dispositivos, controle de dispositivos e assinatura de alterações de estado.

3.4.5.1 API Device Architecture – ADA

A abstração lógica precisa dos dispositivos de IoT para garantir a uniformidade no acesso a recursos é chamada de ADA (*API Device Architecture*), que é estruturada por herança da UPnP Device Architecture [80]. Assim, o ADA é composto por sub-entidades UPnP (dispositivos, serviços, ações, variáveis de estado e argumentos) e, adicionalmente, introduz uma sexta entidade, o assim chamado controlador.

No UPnP, um dispositivo fornece serviços aos usuários. Cada serviço é entregue por meio de ações. As ações possuem parâmetros, chamados argumentos, para orientar como um estado de dispositivo deve ser alterado. Um serviço tem variáveis de estado que são modificadas pela execução de ações e são representadas em interações como argumentos de entrada e saída.

Por exemplo, considerando uma abstração inteligente de lâmpada *dimmer* de acordo com o UPnP e o ADA herdado, a representação poderia ter o *dimmer* como um dispositivo de “brilho de controle”. Este serviço pode ter ações como: “aumentar o brilho”, “reduzir o brilho” e “definir o brilho para um valor especificado”. A ação “definir brilho para um valor especificado” pode ter dois argumentos de entrada: “novo valor de brilho em Lumens” e “tempo de espera antes de alterar o brilho”. O argumento de entrada “novo valor de brilho em Lumens” mudaria a variável de estado de serviço “brilho”. A ação “definir brilho para um valor especificado” pode retornar um argumento de saída com o novo “valor com que o brilho foi definido”, ou seja, o valor da variável de estado “brilho” após a ação ter sido executada.

Para ajudar a abranger objetos simples legados, transformados em um objeto inteligente através da mediação de microcontroladores, a ADA introduz o controlador, uma nova entidade que estende a arquitetura de dispositivos UPnP. Para obter melhor utilização de recursos, pressupõe-se que um controlador possa conter mais de um dispositivo. Embora a arquitetura de dispositivo

UPnP suporte originalmente um recurso que poderia resolver esse problema, na forma de dispositivos incorporados em um dispositivo raiz, o recurso no ADA é mais didático, formal e claro para distinguir o caso de dispositivos físicos multifuncionais, como uma impressora e um *scanner* dentro do mesmo dispositivo físico, do caso em que vários dispositivos fisicamente separados são gerenciados por um microcontrolador, como o conjunto de aparelhos em uma sala controlada por um Raspberry Pi. Com suas subentidades e seus relacionamentos, a ADA possui métodos bem definidos para pesquisar, controlar e monitorar o estado de todos os dispositivos englobados, constituindo uma solução para lidar com a diversidade de dispositivos de IoT, trazendo uniformidade para acessar seus recursos.

3.4.5.2 API Abstract Model

Para aumentar a escalabilidade em configurações de IoT, é interessante usar REST e SOAP, pois eles podem trocar informações por HTTP, um protocolo de transferência que resulta no baixo acoplamento entre cliente e servidor, uma característica que contribui para a escalabilidade desejada. Essa opção reforça o princípio de modelar a API para manipular uniformemente objetos IoT, sejam eles objetos inteligentes ou legados, por meio do ADA apresentado. Além disso, a API é projetada para integrar um módulo de segurança que possui componentes em todas as camadas da API, fornecendo proteção às informações processadas desde o momento em que ela entra na API. Esta tese está focada nas funcionalidades dos serviços de interoperabilidade, supondo que este tipo de serviço de segurança esteja implementado em cada instância IoT participante da rede P2P.

Em seguida, as solicitações enviadas para a API são tratadas por todos os seus módulos de camada. Cada camada tem componentes dedicados a manipular e fornecer informações específicas sobre cada entidade presente no ADA que possui uma entidade controladora, o roteador de solicitação, para redirecionar de forma transparente a solicitação do cliente para o componente apropriado. Portanto, a API permite o gerenciamento separado de cada entidade apresentada nas arquiteturas de IoT, por meio de um padrão de comunicação uniforme.

A apresentação, a lógica de negócios e as camadas de acesso a dados foram projetadas de acordo com os padrões tradicionais de sistemas da Web [81], embora apresentem algumas diferenças fundamentais. Por exemplo, a camada de apresentação da Web geralmente consiste em componentes que fornecem uma ponte comum na lógica de negócios principal encapsulada na camada de negócios e geralmente gerencia a interação do usuário com o sistema. No ADA, a camada de apresentação também é uma ponte para a camada de lógica de negócios, mas o sistema não lida com as interações do usuário. Em vez disso, permite que aplicativos clientes enviem várias solicitações de objetos inteligentes em uma arquitetura de IoT. As características e os recursos das camadas do ADA e seus respectivos componentes são os seguintes:

- Camada de apresentação (PL): esta camada contém os componentes que implementam e exibem a interface de solicitações do cliente e gerenciam as solicitações do cliente, em

todo o REST e SOAP. Essa camada inclui componentes para controlar solicitações de clientes e rotear as solicitações enviadas, sendo composta por dois módulos lógicos principais chamados de roteador de solicitação (RR) e módulo de manipulador (HM). A principal função do módulo RR é redirecionar as solicitações do cliente para os componentes do HM. O roteador de solicitação coleta todas as informações na solicitação REST ou SOAP, cria um objeto preenchido com os parâmetros coletados e envia esse objeto para o componente que manipula a entidade de destino. O módulo do manipulador recebe um objeto de rota enviado pelo RR e envia uma solicitação para a camada de negócios. Especificamente, um componente HM (manipulador de solicitação) envia o tipo de solicitação REST/SOAP (GET, POST, PUT, DELETE), o identificador de objeto de rota e os parâmetros da solicitação para o componente correspondente no módulo de controle, este último sendo responsável por pesquisar o objeto de sessão solicitado no módulo de execução. Vale ressaltar que o módulo manipulador também valida o objeto de rota submetido, como um dos recursos de segurança da API.

- Camada de lógica de negócios (BLL): essa camada implementa a funcionalidade principal da API e encapsula a lógica de negócios relevante. O BLL compreende dois módulos, ou seja, controle e execução, que juntos gerenciam objetos inteligentes ativos na infraestrutura de IoT. É importante ressaltar que, ao contrário dos serviços da Web tradicionais, a camada de lógica de negócios da API não expõe as interfaces de serviço, que só podem ser acessadas por meio da camada de apresentação. O módulo de controle BLL (CM) compreende componentes controladores que podem enviar comandos para alterar o estado dos objetos de sessão, bem como para recuperar informações desses objetos. O CM também pode se comunicar com a camada de acesso a dados se as informações solicitadas não puderem ser entregues por nenhum objeto de sessão. O módulo de execução BLL (EM) gerencia os objetos de sessão atualmente sendo usados na rede IoT. Objetos de sessão podem ser definidos como instâncias de entidades UPnP que armazenam informações necessárias para algum serviço de cliente específico. A principal vantagem de usar objetos de sessão é recuperar informações sobre os dispositivos e serviços sem a necessidade de consultar o banco de dados, permitindo acesso mais rápido às informações solicitadas. Assim, cada componente EM responde às solicitações feitas pelo módulo de controle. Se algumas informações não estiverem disponíveis no módulo de execução, uma solicitação será enviada à camada de acesso a dados e um novo objeto de sessão será criado.

- Data Access Layer (DAL): essa camada fornece acesso a dados hospedados no sistema. O DAL fornece interfaces que podem ser usadas pelos componentes da camada de negócios. A camada de acesso a dados tem apenas um módulo, chamado de módulo mapeador (MM), cuja função é mapear todos os dispositivos e serviços que podem ser solicitados por um cliente. Assim, as entidades presentes neste módulo representam todos os objetos inteligentes que podem ser usados para compor uma rede IoT e que podem estar dentro de um *middleware* IoT. Este módulo também inclui um componente auxiliar de dados que abstrai a lógica necessária para acessar os armazenamentos de dados subjacentes. Ele centraliza

as funções comuns de acesso a dados, a fim de tornar o sistema mais fácil de configurar e manter. O objetivo de realizar esse mapeamento por meio de ajudantes de dados é recuperar dados solicitados de qualquer tipo de armazenamento de dados (como bancos de dados tradicionais, bancos de dados NoSQL, arquivos de texto, XML e JSON, etc.) e convertê-los em um local bem conhecido. objeto de modelo definido.

3.4.5.3 Fluxos de processamento dentro da API REST/SOAP para serviços IoT

Com base na representação lógica da API descrita de objetos inteligentes e em sua estrutura modular interna, a API permite rotas para solicitações REST e SOAP e executa o fluxo de processo correspondente.

O design suporta dezessete fluxos de processamento de solicitações, que são classificados da seguinte maneira: catorze dessas solicitações são destinadas a recuperar informações sobre entidades ADA e um dos fluxos restantes destina-se a emitir comandos de controle para dispositivos, enquanto os outros dois destinam-se a permitir a subscrição de pedidos para receber notificações de novos valores assumidos pelas variáveis de estado dos dispositivos.

Essa assinatura permite que os aplicativos acessem em tempo real os estados dos dispositivos em um sistema de estilo PUSH. A abordagem de assinatura REST requer a especificação direta do URI e seus respectivos métodos e parâmetros, enquanto o SOAP precisa dessas mesmas informações incorporadas em envelopes XML.

3.5 Aspectos de Mobilidade, Geolocalização e Tempo

Já discutimos na presente tese que o modelo proposto, P2PIoT, é organizado e dotado de funcionalidades que permitem ações de interoperação no conjunto de instâncias de IoT federadas em determinado momento, seja a federação decorrente da mobilidade, seja por determinação administrativa.

Nesse contexto, foi discutida a correlação entre a representação abstrata dos recursos nas instâncias de IoT e dos respectivos fluxos de programas e o tempo necessário para executar os serviços, um tema que influencia atividades de interoperação durante a fusão de instâncias de IoT e depois na IoT federada, especialmente se a mobilidade for considerada e a geolocalização for usada por instâncias IoT como meio de detectar outras instâncias de IoT a serem mescladas. A avaliação experimental feita em [61] mostra que a representação abstrata de entidades como serviços REST oferece tempo de resposta mais rápido e consome menos recursos do que serviços SOAP semelhantes.

Por outro lado, vale notar que, independente da causa que leva à junção das instâncias de IoT na estrutura virtual P2P, o desempenho de todas as funções de interoperação que são suportadas depende dos tempos de execução dos softwares envolvidos, bem como de características temporais das comunicações interativas entre os módulos de software, com particular importância ao

fator de latência das comunicações.

Por tal razão, tais temas são discutidos nas seguintes subseções da presente tese.

3.5.1 Papel Geral da Geolocalização na Federação e Particionamento de Instâncias de IoT

De fato, a federação por determinação administrativa, ou seja, por comando executado manualmente nos *middlewares* envolvidos, pode se referir a recursos (dispositivos, aplicações, módulos de *middleware*) que podem se encontrar em qualquer ponto do espaço de endereçamento da Internet, desde que o respectivo endereço IP tenha sido mapeado no Módulo de Serviços de Busca P2P utilizado no Modelo P2PIoT. Nesse caso, o cálculo da latência dependerá de alguma forma de geolocalização dos endereços IP, possivelmente com apoio de um sistema de geolocalização global, haja vista que a possibilidade de consultas a bases de dados de coordenadas geodésicas é muito suscetível ao erro.

Já a federação causada pela mobilidade das instâncias de IoT depende diretamente de geolocalização, para que as instâncias de IoT envolvidas possam ter a percepção de presença das demais instâncias, o que pode ser um processo automático vinculado à função de interoperação *merge*.

Por outro lado, é importante que as soluções P2P tenham ciência das partições de rede, haja visto que estas partições podem ocorrer inesperadamente a qualquer momento, seja devido a falhas na infraestrutura de comunicações ou no sistema de roteamento que podem interromper partes da Internet, o que afeta até mesmo redes fixas sobre as quais opera o P2PIoT, seja devido à mobilidade das instâncias que estejam federadas em determinado momento e em seguida se afastem fisicamente.

3.5.2 Estabilização das Instâncias IoT pós Particionamento

No caso de ocorrer um evento de particionamento de rede, os participantes de uma sobreposição P2P interrompida devem ser capazes de manter a interoperação nas regiões geográficas remanescentes. Em cada nova divisão, a rede de sobreposição P2P deve ser mantida e deve estabilizar rapidamente, o que é um requisito à função de interoperação *split*. Além disso, uma vez que a conectividade da rede subjacente seja restabelecida, as várias sobreposições devem detectar a presença das demais e voltar se fundir global novamente, o que novamente é requisito à função de interoperação *merge*.

Esta tese, inspirada na sistemática da rede P2P Chord, e sem perder a generalização, focaliza a proposta P2PIoT em redes de sobreposição baseadas em anel que são caracterizadas por mapear nós para pontos de um espaço identificador circular e por conectar pares de acordo com sua posição neste espaço identificador.

De acordo com [69], enquanto a separação de nós únicos em redes P2P é um desafio comum que os pesquisadores abordam regularmente, a separação súbita de grupos inteiros de nós raramente é considerada, embora isso ocorra quando um conjunto de nós é separado de uma sobreposição P2P existente. Isso é exatamente o que acontece com os nós correspondentes aos recursos de uma instância da IoT quando essa instância se afasta de uma IoT federada.

De fato, para a operação de divisão (*split*), se nenhum outro mecanismo de controle estiver em vigor quando duas instâncias estiverem isoladas uma da outra e nenhum roteamento entre elas for possível, pode ocorrer que mais de dois anéis (Chord) surjam após um certo período de estabilização. Embora os nós de cada anel individual na mesma instância possam estar conectados no nível da rede, eles podem estar incoerentes na sobreposição.

Assim, durante o período de isolamento, apenas os nós conectados são levados para a formação de um grupo, que é representado no mesmo conjunto de entradas do anel Chord e seus ponteiros sucessores ou predecessores. No entanto, também não há garantia de que, se as instâncias separadas forem reconectadas novamente no nível da rede e ambas as instâncias forem capazes de rotear entre si, as duas instâncias convergirão para um anel comum. A razão para esse comportamento é que nenhum nó em qualquer sobreposição separada mantém qualquer informação de roteamento em direção a pelo menos um nó de contato na outra sobreposição separada. Consequentemente, o roteamento em uma sobreposição que reintegra ambas as instâncias como participantes pode se tornar impossível e as duas instâncias permanecerão independentes uma da outra.

Para evitar tais problemas, relacionados a divisões e mesclagens sucessivas, especialmente quando o gerenciamento de mobilidade é automático, no Modelo P2PIoT o controle dessas operações é de responsabilidade exclusiva do **Módulo de Serviços de Interoperação P2P**, ou seja, essas operações não exigem nenhuma ação dos dispositivos IoT e aplicações. Isso constitui a medida de controle adicional que é programada nos serviços de interoperabilidade *merge* e *split* fornecidos no P2PIoT.

Como observado por [69], é interessante destacar que a funcionalidade de mesclar outros anéis não é fornecida no Chord original e em muitos sistemas de sobreposição relacionados. Mas, essa funcionalidade é altamente desejada, pois permitiria o uso de sobreposições em uma ampla gama de cenários, incluindo aquele que é a questão central desta tese, ou seja, a necessidade de dar suporte à interoperação entre diferentes instâncias de IoT.

3.5.3 Geolocalização e Cálculo de Proximidade Precedentes à Mesclagem de Instâncias de IoT

Para a fusão de várias sobreposições P2P não conectadas, primeiramente, é necessário que pelo menos um nó de *middleware* de instância da IoT em uma sobreposição (o iniciador) entre em contato com outro nó de *middleware* de instância da IoT de outra sobreposição (nó de contato) para iniciar o algoritmo de mesclagem. A decisão de realizar o contato depende de uma decisão

administrativa ou, no caso automatizado, de algumas informações provenientes de um sistema de posicionamento que afirma que as instâncias de IoT foram movidas para compartilhar a mesma área geográfica, ou seja, estão em um estado de proximidade. Um artigo relacionado anterior [82] discute os aspectos de convergência e desempenho deste último processo de detecção de proximidade. Em todo caso, a realização de tais cálculos tem como efeito adicional a obtenção do atraso nas comunicações entre as instâncias envolvidas.

Em P2PIoT, o algoritmo proposto é projetado para fusão confiável na presença de várias sobreposições P2P paralelas, usando diretamente a operação de junção do Chord, com o objetivo de reduzir a sobrecarga de fusão em termos de consumo de largura de banda e tráfego, conforme descrito a seguir. Essa sobrecarga está relacionada apenas ao número de construções de sobreposição, ou seja, instâncias de IoT a serem mescladas na rede e não ao número de nós que representam recursos internos de cada uma dessas instâncias de IoT.

Na solução adotada no P2PIoT, informações locais nos nós que contêm o *middleware* das instâncias de diferentes anéis Chord são suficientes para mesclar esses anéis. Cabe a um controlador de *middleware* de instância atuar como um nó de iniciador que toma conhecimento de qualquer outro nó no outro anel e, em seguida, poderá iniciar uma pesquisa para seu sucessor, que é o *middleware* IoT no outro anel.

Posteriormente, o nó iniciador inicia o procedimento de mesclagem enviando suas informações de roteamento originais para o sucessor e solicita ao sucessor alternativo para seu antecessor, sua lista de sucessores, sua tabela de ponteiros para a parte do espaço identificador pelo qual será responsável após o fusão. Ao receber as informações do iniciador, o sucessor responde enviando as informações solicitadas ao iniciador.

Simultaneamente, o sucessor encaminha o processo de mesclagem para outros nós que representam instâncias previamente mescladas e o processo reitera com essas outras instâncias de *middleware*. O processo de mesclagem é repetido por cada nó que contém uma instância de *middleware* até que a solicitação de fusão chegue de volta ao nó do iniciador.

Finalmente, o nó iniciador combina sua lista de sucessores e sua tabela de ponteiros com as informações obtidas do sucessor alternativo. Nessa solução, o *middleware* controlador de uma instância de IoT representa todos os membros dessa instância, uma situação análoga à representação de um grupo de confiança em que a representação se refere ao sistema inteiro ao invés de a uma entidade particular, conforme [83].

O tempo de execução da fusão, que consiste em uma operação de junção do Chord, depende do número de nós a serem unidos e, no pior dos casos, dos atrasos relacionados a informações provenientes de um sistema de posicionamento que permite declarar que as instâncias de IoT foram movidas para compartilhar a mesma área geográfica, ou seja, que tais instâncias estão em um estado de proximidade.

Para avaliar a qualidade desses algoritmos de particionamento e mesclagem, também é necessário quantificar a sobrecarga em termos de consumo de mensagens e largura de banda. Idealmente, a sobrecarga de tráfego de uma abordagem de fusão deve ser em relação ao número

de construções na rede subjacente. A sobrecarga deve ser mínima, mas espera-se que aumente linearmente, enquanto mais anéis implicam em mais sobrecarga devida à pesquisa de contato/-proximidade e consequente processo de fusão das instâncias de IoT envolvidas.

3.5.4 Repetitividade do Cálculo de Proximidade em Situação de Mobilidade das Instâncias de IoT

Na verdade, a pesquisa de contato é um processo que deve ser repetitivo para detectar tanto novas instâncias de IoT nas proximidades, mas também para detectar a ocorrência de afastamento entre instâncias de modo a interromper interoperações que envolvam recursos das instâncias reunidas e sobretudo para acionar a operação de *split* dessas instâncias que se afastam.

Desse modo, há uma influência geral dos atrasos do sistema de posicionamento nos diversos serviços de interoperação propostos nesta tese, caso a mobilidade seja considerada e gerida automaticamente. Por essa razão, vale discutir os estudos feitos correlatos a esta tese concernentes ao uso de sistemas de posicionamento ou geolocalização.

3.5.5 Interrelações entre Atraso, Precisão e Complexidade Algorítmica na Geolocalização

No sentido de discutir a utilização do posicionamento na execução do modelo proposto nesta tese, vale citar o artigo precedente e diretamente correlacionado à temática abordada [14], artigo este no qual é tratada e discutida a questão do atraso e da precisão dos Sistemas Globais de Navegação por Satélite (em inglês, *Global Navigation Satellite Systems* – GNSS), como GPS, Galileo, GLONASS e Beidou. Tal artigo argumenta que os GNSS são vitais para diversas aplicações, como aviação civil, defesa, marcação do tempo e sincronização de redes críticas, cabendo notar que instâncias de IoT são de grande aplicação em tais domínios.

No citado artigo [14], para calcular a posição do receptor GNSS na superfície da Terra, este receptor executa o cálculo de alcance usando sinais de linha de visão (em inglês, *line-of-sight* – LOS) de pelo menos quatro satélites.

Em função das características estruturais do ambiente de propagação, os reflexos dos sinais de satélite em árvores, lâmpadas e edifícios geram componentes de múltiplos caminhos. A superposição dos sinais de LOS e componentes de multipercurso no receptor degrada a estimativa de tempo de atraso e, consequentemente, a estimativa de posição.

A fim de mitigar o efeito dos componentes do multipercurso, o artigo considera os receptores GNSS equipados com matrizes de antenas e propõe uma nova abordagem para cálculos de arranjos de antenas.

Comparada com abordagens de ponta existentes, a abordagem proposta tem melhor o desempenho para o caso de sinais altamente correlacionados e resulta em estimativas de alta resolução

para o tempo de atraso, embora seja computacionalmente mais cara e requeira uma implementação mais complexa, sendo também sensível a pequenas diferenças do ângulo de chegada.

3.5.6 Automação da Gerência de Mobilidade das Instâncias de IoT

Vale discutir também a situação em que o gerenciamento de mobilidade é automático, como é o caso quando a instância de IoT é suportada por redes ad hoc, espontâneas ou auto-organizadas. Existe então a possibilidade de a instância de IoT apelar para seus protocolos de controle de acesso e roteamento de rede para ajudar com os processos de mesclar e dividir.

No contexto desses protocolos de suporte, existem problemas relacionados à mesclagem e divisão que são análogos aos percebidos no nível do sistema distribuído da IoT. A referência [84] observa que, como consequência da mobilidade, é provável que as redes espontâneas se sobreponham espacialmente e isso permita a comunicação entre as tais redes que até então estavam separadas, mas há pelo menos dois requisitos para que essas comunicações entrem em vigor.

Primeiro, a fusão só deve ser disparada quando as redes espontâneas sobrepostas detectarem a presença uma da outra. Segundo, a região sobreposta - ou seja, a região através da qual todas as comunicações passarão - deve ser claramente delimitada.

Em seguida, os protocolos de roteamento ad hoc devem se adaptar automaticamente às condições da rede e prosseguir com a mesclagem. Esse mesmo artigo [84] aponta que a necessidade de mesclar é geralmente detectada por nós pertencentes a diferentes redes através da troca de identificadores de rede, que supostamente são números aleatórios atribuídos a redes diferentes, já que os identificadores devem ser únicos para uma rede detectar que a fusão está prestes a ocorrer. Outra possibilidade é que os identificadores podem vir de um protocolo de configuração ad hoc [85].

Em todo caso, a troca de mensagens de detecção de mesclagem é realizada através de mensagens de descoberta típicas usadas em redes espontâneas, por exemplo: HELLO, e assumindo que os nós sabem suas coordenadas através de algum sistema de posicionamento e então são capazes de calcular e informar mutuamente seus respectivos vetores de velocidade. Assim, temos os mesmos problemas de cálculos de posicionamento para os nós e sua proximidade, conforme discutido acima.

3.5.7 Pragmática do Modelo Proposto P2PIoT quanto a Desempenho dos Controles de Geolocalização e Proximidade

Considerando as duas situações descritas, tanto no referente à latência dos sinais de satélite quanto à complexidade do cálculo de posição, bem como dos cálculos consequentes de proximidade entre duas ou mais instâncias de IoT, conclui-se que o atraso na execução do software P2PIoT tem o requisito de ser muito menor do que a soma do cálculo provido pelo serviço de localização e do cálculo da proximidade, para que as operações na IoT tenham um tempo máximo

de realização conhecido e limitado.

Entretanto, o tema da determinação dos atrasos envolvidos na interoperação entre sistemas distribuídos heterogêneos, como é o caso da interoperação entre instâncias de IoT, é de difícil abordagem de forma analítica, em função da imprevisibilidade dos entes envolvidos, sistemas operacionais, módulos de software distribuídos, sistemas de comunicação, redes de formato indefinido *a priori*, etc. Por essa razão, a avaliação dos tempos relativos à proposta desta tese é realizada de forma experimental, com medições de desempenho que fazem parte do processo de validação da proposta, conforme apresentado no capítulo 4.

Capítulo 4

Validação e Discussão dos Resultados

Com o objetivo de validar a proposta do Modelo de Interoperação para Internet das Coisas P2PIoT, já descrito nesta tese, bem como de gerar parâmetros de avaliação para cada módulo constituinte desse modelo, foi desenvolvido um protótipo para demonstrar as funcionalidades e obter métricas de funcionamento em cenários representativos de instâncias de redes IoT funcionando de forma paralela e simultânea e interoperando nessas circunstâncias.

Para efeito dos testes de validação, o protótipo desenvolvido teve como requisitos apresentar pelo menos duas instâncias de redes IoT de forma que o *middleware* em cada uma dessas instâncias receba as bibliotecas com as funções de interoperação do P2PIoT, adaptando-se assim cada instância a um módulo de sistema busca P2P que, sendo comum às duas instâncias, permita ativar os serviços de interoperação de IoT que devem ser providos segundo o modelo P2PIoT.

Para realizar tal protótipo, o ponto de partida foi o *middleware* IoT desenvolvido no Laboratório UnB Internet of Things – UIoT, da Universidade de Brasília, tendo em vista a possibilidade de reutilizar o código-fonte disponível e cujos princípios, arquitetura e funcionalidades vêm sendo concebidos e validados na forma de publicações como [86], [61] e [10]. Em tais referências, a descrição da arquitetura e dos serviços do *middleware* UIoT apresenta especificamente o conjunto de procedimentos, recursos e serviços para que ocorra a interoperação interna a uma instância de IoT, ou seja, entre os dispositivos, o *middleware* e as aplicações de IoT, provendo as identificações necessárias, as abstrações de serviços de software e os fluxos de informação para que a interoperabilidade ocorra de forma correta no interior da instância. Assim, o trabalho de prototipação do Modelo P2PIoT consistiu sobretudo de dotar o UIoT de capacidades para interoperação entre diferentes instâncias.

Vale observar que o *middleware* UIoT provê uma estrutura de módulos definidos de um *middleware* principal, mas também de um módulo de *gateway* UIoT que opera separado fisicamente do módulo principal, mas com a mesma lógica de funcionamento. Cada *gateway* UIoT na verdade é um módulo compactado do próprio *middleware* principal UIoT e se destina a operar na borda (*edge*) das instâncias de IoT, ou seja, objetivando ficar mais perto dos dispositivos de IoT nas situações em que ocorre ou concentração de dispositivos e/ou uma grande quantidade de protocolos

de acesso que perturbariam o funcionamento do *middleware* pincipal ou exigiriam mais complexidade da rede de comunicação, para garantir a conectividade dos dispositivos e a conversão de protocolos.

Entretanto, o módulo de maior importância para a validação desta tese é o próprio *middleware* principal, pois é neste módulo que são implementados os componentes do Modelo P2PIoT, haja vista a discussão apresentada no Capítulo 3 acerca da escolha do componente de acoplamento do módulo de interoperação do P2PIoT com cada instância de IoT que necessite de funcionalidades de interoperação com outras instâncias. Em consequência dessa escolha, é em torno do *middleware* principal do UIoT que são organizados os experimentos para validação desta tese.

Assim, no desenvolvimento do protótipo de validação do Modelo P2PIoT, foi acoplado ao *middleware* UIoT um módulo de software implementado com as funções de interoperabilidade P2PIoT, ou seja, aquelas que realizam os serviços de *merge*, *split*, *copy*, *move*, *delete*, diretamente entre pares de instâncias de IoT. Também foi feita a integração do *middleware* UIoT com o sistema P2P Chord, para que tais funções possam ter alcance entre diferentes instâncias de IoT, cada uma delas executando o protótipo UIoT+P2PIoT.

Com tal protótipo desenvolvido e operante, foram organizados os testes de validação da tese, consistindo de, em um ambiente controlado, ou seja em rede fechada sem acesso à Internet, colocar em execução duas instâncias de rede IoT cada uma com os softwares componentes do UIoT+P2PIoT, com suas funcionalidades operantes. Tais componentes e funcionalidades foram exaustivamente testados em dois principais cenários de validação da proposta da tese, sendo o primeiro cenário concebido para que validasse as funcionalidades do P2PIoT em todas os serviços de interoperação previstos. Já o segundo cenário visa a avaliação da performance, comparativamente com um ambiente centralizado de IoT, ou seja um ambiente em que a interoperação é feita com a intermediação de sistema central ao invés de ser diretamente entre as instâncias de IoT utilizando o modelo P2PIoT. Assim, o segundo cenário envolve a configuração de duas instâncias do UIoT em uma nuvem privada, além da configuração de duas instâncias UIoT+P2PIoT em rede P2P utilizando o sistema Chord.

Este capítulo descreve o ambiente de desenvolvimento de software usado na construção do protótipo e detalha os cenários de validação, discutindo os resultados de testes em cada cenário. Além do ambiente de desenvolvimento do protótipo, descreve-se a arquitetura do protótipo e, para os cenários de testes, são apresentados os resultados obtidos com essa implementação do Modelo P2PIoT.

4.1 Ambiente de Desenvolvimento do Protótipo P2PIoT

Para atender aos requisitos da metodologia adotada na tese, foi necessário o desenvolvimento e a implementação em laboratório de um protótipo para realizar cenários de validação destinados a testes específicos que resultassem na verificação do funcionamento do modelo proposto e obtenção das correspondentes métricas de performance.

Para tanto, é necessário que o protótipo implementado seja representativo do funcionamento dos dispositivos e do *middleware* de IoT em situação que envolva pelo menos duas instâncias de rede IoT em interoperação par-a-par, de modo a gerar resultados referentes ao Modelo P2PIoT que mostrem o efetivo funcionamento dos serviços de interoperação e sirvam à comparação com um modelo de interoperação entre as citadas instâncias por meio de um serviço centralizado.

Para atender a tais requisitos, o protótipo P2PIoT foi desenvolvido utilizando a linguagem de programação Python, com auxilio de alguns módulos de conveniências dessa linguagem, como o PyLab (plotagem) e o Flask (construção de aplicações *web*). Além disso, foram utilizados alguns *plugins*, em sua maioria com licenças gratuitas ou de código aberto, para interações em Python. A escolha de tais módulos foi determinada pelos problemas específicos nos desenvolvimento do protótipo, que serão mostrados com maior detalhe no item 4.4 deste capítulo.

As subseções seguintes tratam em específico de cada um elementos utilizados para desenvolvimento do protótipo e sua implementação em ambiente controlado de testes, este utilizado nos cenários de validação da proposta da tese.

4.1.1 Python

O uso de Python [87] como linguagem de programação deve-se a vários motivos, cabendo principalmente citar alguns pontos fortes dessa linguagem que influenciaram na implementação do protótipo de validação deste trabalho.

O primeiro ponto forte é que é uma linguagem altamente portável. Isto quer dizer que o desenvolvimento de eventuais classes ou soluções podem ser migrado para outras plataformas sem grande esforço.

Outro ponto importante é que se trata de uma linguagem de programação dinâmica que pode ser utilizada em diversos tipos de aplicações (*Web Application*, *Win32 App*, redes, aplicações científicas, aplicações móveis, entre outras). Além disso, é uma linguagem de fácil integração com outras linguagens e ferramentas (C++, java, .NET, C, MatLab e Delphi), possuindo uma extensa biblioteca para tais fins.

A linguagem Python foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade da execução, tendo uma sintaxe concisa e clara. Além disso, conta com os recursos poderosos de sua biblioteca padrão e também com módulos e *frameworks* desenvolvidos por terceiros. É assim uma linguagem e ambiente de desenvolvimento bastante adequada para efeito da validação desta tese, por se adaptar à situação com múltiplos módulos de software distribuídos e paralelos que devem atuar de forma coordenada.

A versão utilizada nesta implementação foi a Python 3.7.3 com os módulos ou *plugins* usados no desenvolvimento do protótipo que são descritos nas subseções a seguir.

4.1.2 Eclipse e PyDev

O Eclipse é uma plataforma livre usada para ambiente integrado de desenvolvimento (*Integrated Development Environment* - IDE) de projetos em diferentes linguagens de programação para distintos sistemas operacionais. Atualmente, suporta linguagens como C/C++, COBOL e Java, sendo bastante usada em aplicações *desktop*. Tal IDE é extensível através o uso de vários *plugins*.

Na validação desta tese, é usado o *plugin* PyDev que permite aos usuários utilizar o Eclipse para desenvolver aplicações em Python e Jpython. Esse *plugin*, que é descrito detalhadamente na referência [87], permite o desenvolvimento de aplicativos Python para *desktops*, possuindo vários recursos como auto-complementação do texto, detecção de erros, entre outros que podem ajudar bastante no desenvolvimento de aplicações, especialmente por causa do *HighLighting* automático e da detecção de erros comuns da sintaxe Python.

O motivo principal da utilização do PyDev Release 7.2.1 para o protótipo P2PIoT foi a sua facilidade de interação com o software Eclipse, o que ajuda sobremaneira a construção de aplicações em Python.

4.1.3 WxPython

O wxPython [88] é uma API de biblioteca wxWidgets [89] que por sua vez é uma ferramenta utilizada na construção da interface com o usuário, o que permite que as aplicações sejam mais facilmente portáveis e que tenham a aparência de uma aplicação nativa do sistema operacional Windows. Além disso, permite escrever interfaces usuário gráficas multi-plataformas no estilo de pyQT ou pyGTK [89].

Além de ser o componente de software que viabiliza a interação com o usuário, sendo um utilitário para criação de *widgets* multi-plataforma, possuindo uma biblioteca com elementos básicos para a construção de interfaces gráficas para o usuário, o wxWidgets possibilita conexões com bancos de dados ODBC e conexões de rede via *sockets*. Permite que desenvolvedores criem aplicações para sistemas operacionais Win32, Mac, entre outros, podendo ser usado a partir de linguagens como C++, Python, Perl, C# e .NET.

O wxPython é utilizado no protótipo P2PIoT com o objetivo da criação de uma interface com o usuário.

4.1.4 Flask

Flask [90] é um pequeno *framework web* escrito em Python e baseado na biblioteca WSGI Werkzeug e na biblioteca de Jinja2. Flask está disponível sob os termos da Licença BSD.

Flask tem a flexibilidade da linguagem de programação Python e provê um modelo simples para desenvolvimento web. Uma vez importando no Python, Flask pode ser usado para economi-

zar tempo construindo aplicações web. É chamado de *microframework* porque mantém um núcleo simples, mas extensível. Não provê abstração de banco de dados, validação de formulários, ou qualquer outro componente, mas bibliotecas de terceiros existem para prover as funcionalidade complementares. Assim, Flask suporta extensões capazes de adicionar tais funcionalidades na aplicação final, o que simplifica o *framework* e torna sua curva de aprendizado mais suave.

4.1.5 Pylab

O Pylab é um módulo da linguagem Python que permite gerar gráficos de duas dimensões de excelente qualidade, permitindo edição interativa e animações, provendo inúmeros tipos de gráficos diferentes, anotações em sintaxe Latex e salvamento das imagens geradas em diversos formatos diferentes. A sintaxe de criação e manipulação das imagens é similar à do software comercial Matlab, mas provendo muito mais capacidades, além de uma interface baseada em objetos.

O Pylab permite trabalhar com diversos tipos de gráficos diferentes, entre eles: gráficos de funções, múltiplos gráficos, histogramas, funções discretas, torta, barra, etc. Fornece funções para a customização dos gráficos, podendo trabalhar com diversas fontes diferentes, cores, tamanhos de página, e muito mais.

4.1.6 JSON

O JSON é uma notação de dados leve, baseada em texto e transparente à linguagem de programação e ao sistema operacional. É um padrão ECMA utilizado em larga escala por programadores de serviços web assíncronos que utilizam, principalmente, JavaScript para interação de aplicação do lado do cliente.

Uma das vantagens do JASON é a de ser uma notação mais compacta que o XML, causando, respectivamente, menos esforço de programação para conversão em objetos de aplicação e menor uso de banda para transmissão de dados. Por outro lado, possui tipos primitivos e estruturados, neste último caso incluindo os tipos *array* e objeto que são construídos com os tipos primitivos que são *strings*, números, booleanos e nulos. Todo documento JSON tem seu escopo definido como sendo o valor de um tipo estruturado ou primitivo. Cada valor de item estruturado pode conter outros itens estruturados ou primitivos.

4.2 Desenvolvimento do Protótipo

O desenvolvimento do protótipo apresentado neste trabalho partiu do Modelo P2PIoT apresentado no item 3.2 e utilizou as ferramentas descritas no item 4.1, de maneira a resultar em um sistema, que embora não tenha um acabamento e um objetivo comercial, tem como característica

demonstrar com simplicidade o funcionamento de Modelo P2PIoT, permitindo testes de validação em dois importantes cenários que são descritos mais adiante neste capítulo.

A metodologia de desenvolvimento do sistema reuniu métodos e padrões da engenharia de software com programação orientada a objetos e utilização de tecnologias de redes para sistemas distribuídos em várias camadas, permitindo assim a decomposição modular do software, a abstração de dados e serviços e a organização geral com a integração dos diversos módulos produzidos.

A implementação do módulo principal descrito no Modelo P2PIoT foi realizada no Laboratório UIoT da UnB, que então serviu de infraestrutura para demonstração da prova de conceito. O uso dos componentes descritos na seção 4.1 permitiu a modificação do código do *middleware* UIoT, da maneira necessária à implementação dos conceitos de redes P2P e criação de interfaces entre os diversos módulos de software, contando com a integração com as bibliotecas do *middleware* UIoT.

4.3 Descrição do Ambiente de Teste

O objetivo do ambiente de testes é permitir a apresentação dos resultados de validação da proposta P2PIoT. Vale notar que em geral as instâncias de IoT seguem os moldes de ambiente distribuído, com cada instância de IoT contendo pelo menos uma aplicação central que, por intermédio do *middleware*, interage com os dispositivos distribuídos na rede. Esse é o modelo adotado no UIoT que é o *middleware* escolhido para validação da tese. Já o ambiente de P2PIoT segue um modelo P2P pelo qual os servidores respondendo pelo *middleware* de cada instância de IoT clientes de outros servidores similares e vice-versa.

Para atender a essa necessidade de validação do acoplamento entre diferentes instâncias de IOT, o ambiente configurado no Laboratório UIoT foi organizado para representar dois cenários pertinentes à validação da tese.

O cenário de validação 1 destina-se a testes de interoperação P2P implementando as funcionalidades do Modelo P2PIoT, de modo a permitir que sejam feitos testes de (*Backup*, *Restore*, *Delete*) de dispositivos entre duas instâncias de redes IoT que interoperação em modo par-a-par, ou seja, diretamente sem intervenção de um mediador central.

Já o cenário 2 destina-se a testes de desempenho realizados em cada transação, permitindo comparar o Modelo P2PIoT proposto e a alternativa de realizar interoperação por intermédio de uma entidade centralizadora.

Em ambos os cenários, para validar o protótipo, os cenários de teste são descritos com o ambiente necessário para o desenvolvimento e apresentação dos resultados. Além disso, são apresentados os critérios para a avaliação destes resultados.

Tais cenários levam em consideração os trabalhos correlacionados relativamente a ambientes de IoT, conforme Capítulo 2, de modo a permitir a realização de testes que pudessem comprovar as funcionalidades de interoperação atendendo ao requisito de adaptação para vários tipos de

dispositivos em vários tipos de *middleware* IoT. Para os cenários de validação escolhidos nesta tese, foi então necessária a criação de um ambiente com os dispositivos devidamente configurados de modo a refletir uma situação realista de interoperação entre instâncias heterogêneas de IoT.

4.3.1 Descrição do Hardware Utilizado

O ambiente utilizado para as simulações do trabalho é composto por 1 computador do tipo desktop com Processador Intel Core-I5 8400 2.8Ghz LGA-1151-G8 9MB Cache, com 8GB de memória RAM, com um espaço em disco de 480GB SSD e com uma placa de vídeo do tipo Off Board com 256Mb, responsável pelo ambiente de codificação e simulação dos dispositivos a serem conectados na rede.

Para o ambiente de aplicação executando o *middleware* utilizamos um servidor virtualizado com 32GB de memoria RAM com um espaço de 200Gb, disponível também para armazenamento dos dados, além de ser responsável pela implementação e controle de fluxo e segurança do *middleware*.

Além disso, foram utilizados 4 Raspberry PI 3 com 128Gb de memória RAM em cartões mini SD, para representação das instâncias de IoT, realizando as operações das funcionalidades propostas no modelo P2PIoT.

Em IoT, as coisas são em geral dispositivos que incorporam computadores digitais embutidos baseados em microprocessadores para funções de controle, análise e comunicação. Em geral, tratam-se de dispositivos em um único circuito integrado que contém um núcleo de processador, memória e periféricos programáveis de entrada e saída.

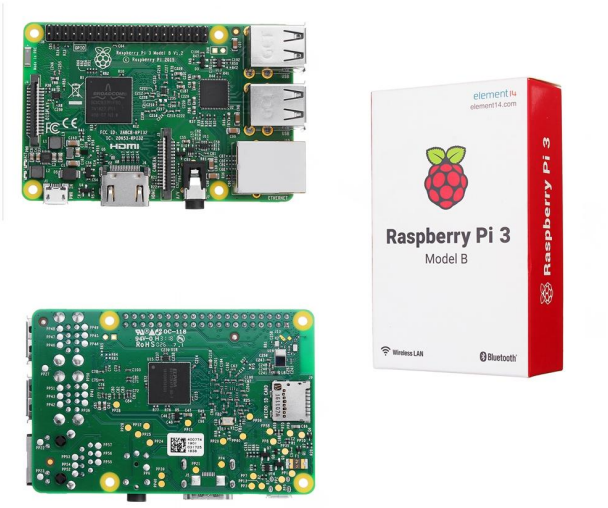


Figura 4.1: Raspberry Pi 3

No caso do desenvolvimento do protótipo desta tese, para contornar a dificuldade com a grande variedade de controladores usados para desenvolver trabalhos em IoT, optou-se por usar

uniformemente o Raspberry Pi3, figura 4.1, haja vista as evidências de que esses controladores são dos mais usados nestes tipos de sistemas e pelo fato de que se trata de um computador completo com sistema operacional, interface com usuário, possibilidade de ligação de periféricos e afins.

Para realização dos testes com os Raspberrys Pi3, foram adotadas as seguintes medidas:

- Foi utilizado um dispositivo Raspberry Pi3 para implementar cada uma das funcionalidades;
- Os dispositivos foram abstraídos por estruturas de objetos no *middleware* UIoT;
- Foi utilizado um dispositivo Raspberry Pi3 para a clonagem do *middleware*.
- Foi utilizado um dispositivo Raspberry Pi3 para execução do Sistema P2P Chord.

4.3.2 Descrição do Software Utilizado

Com relação ao conjunto de softwares, foi utilizado como sistema operacional do ambiente de desenvolvimento o Linux centOS. No lado dos clientes representativos de dispositivos foi utilizado o sistema operacional Linux Ubuntu para Raspberry, enquanto os servidores de aplicação do *middleware* UIoT, com os módulos do P2PIoT, utilizam Linux CentOs.

As informações gerenciadas pelo *middleware*, no formato das abstrações usadas nesse sistema, ficam armazenadas em um banco de dados do UIoT que é alimentado a partir da interação com diversos tipos de objetos/dispositivos inteligentes (*smart objects*), potencialmente dispostos em diversos locais diferentes.

Para que os procedimentos de teste ocorressem de uma melhor maneira para efeito de apresentação, evitamos a variabilidade de dispositivos e adotamos um dispositivo padrão na configuração de testes, isso sem perda de generalidade, haja vista que o *middleware* de cada instância de IoT é responsável pelas abstrações de estruturas de dados e serviços de cada dispositivo particular.

4.3.3 Descrição da Configuração de Rede

Como a IoT é um paradigma onde as coisas são capazes de obter dados ou informações do mundo físico real os dados percebidos por essas coisas são enviados pela Internet, é necessário para efeito de validação da tese a configuração de uma rede controlada, no sentido de permitir a observação do fenômeno de interoperação de instâncias de IoT. Vale notar que as instâncias de IoT se sobrepoem diretamente à Internet, com os dispositivos usando tecnologias sem fio, como por exemplo GPRS, 3G, 4G, Wi-Fi ou através de um *gateway*.

Configuramos assim, para efeito de validação da proposta da tese, uma rede baseada em protocolo TCP/IP, sendo esta confinada no Laboratório UIoT, de modo a constituir um ambiente

de observação da interoperação entre isntâncias de IoT. Tal rede tem uma configuração comum na qual é organizado um *backbone*, interligando os dispositivos da rede por um *patch panel* com *switches* para centralizar as conexões físicas dos dispositivos (SW02 e SW03), estes *switches* mesmos conectados num *switch* central (SW01), ao qual é conectado um roteador para eventuais acessos externos. Tal roteador, por sua vez, está ligado a Internet e segmenta a rede de testes por sub-redes, de forma a agrupar os dispositivos representativos de instâncias de IoT e restringir o acesso não autorizado.

4.3.4 Serviço de Aplicação de *Middleware* de IoT

No contexto de IoT, o *middleware* é o software que serve para aglutinar e dar coerência a diferentes componentes de sistemas operacionais, aplicações e ambientes de hardware e comunicações.

Para tanto, o *middleware* tem duas principais responsabilidades, sendo uma delas traduzir as requisições de aplicações IoT em comandos executáveis e transmite os comandos corretos para os dispositivos alvo. Esse repasse de dados ocorre de um controlador que representa e conversa com aplicações, para um controlador representa e conversa com dispositivos. Para tanto, o *middleware* utiliza suas abstrações lógicas de dispositivos e repassa os comandos entre as entidades físicas, que executarão os comandos no dispositivo alvo. Uma vez finalizado esse processo, o *middleware* retorna uma resposta para a aplicação notificando a conclusão da operação.

Já a segunda rotina do *middleware* consiste em monitorar o estado de um dispositivo e, quando perceber uma alteração, notificar as aplicações desse novo estado. Para isso, as entidades físicas monitoram o estado de um dispositivo e, quando percebem a modificação, comunicam o novo estado de um controlador de dispositivo para um controlador de aplicação, que notificará aplicações. Isso é feito utilizando a comunicação entre entidades físicas, abstração e dispositivos e o gerenciamento de assinantes. Este último bloco é responsável por saber quais aplicações, dentre as que se comunicam através da API do *middleware*, estão interessadas em saber o estado de quais dispositivos e, quando houver alterações, notificar o novo estado de tais dispositivos [91].

Na configuração de testes desta tese, um servidor específico executa o *middleware* do UIoT ao qual foi incorporado o conjunto de módulos do Modelo proposto P2PIoT.

As baterias de teste foram realizadas em um ambiente controlado, ou seja, sem a influência de tráfego e de eventos de uma rede aberta à Internet. Tais baterias de teste são melhor descritas no contexto de cada cenário a seguir apresentado.

4.4 Descrição Geral dos Cenários de Testes

Para efeito de contextualização, os cenários de teste consideram as possibilidades de interoperação mostradas na 4.2, na qual se representam as seguintes entidades e procedimentos:

- Apresentam-se duas instâncias de IoT A e B, cada uma com seu próprio módulo Chord integrado ao UIoT+P2PIoT. Tais instâncias podem se mesclar formando uma federação [A][B], o que é representado pelo procedimento *merge*;
- Na federação [A][B], os módulos UIoT+P2PIoT interoperam compartilhando o módulo Chord resultante da mesclagem. Assim, várias operações de cópia, apagamento e restauração de dispositivos podem ser realizadas, o que é representado pelos procedimentos *copy* e *move*. Na verdade, durante a federação, as instâncias A e B mantêm sua individualidade e apenas se encontram virtualmente na mesma rede de sobreposição Chord;
- A qualquer momento a federação pode ser desfeita, com a separação das instâncias que voltam a sua existência individual, o que é representado pelo procedimento *split*.

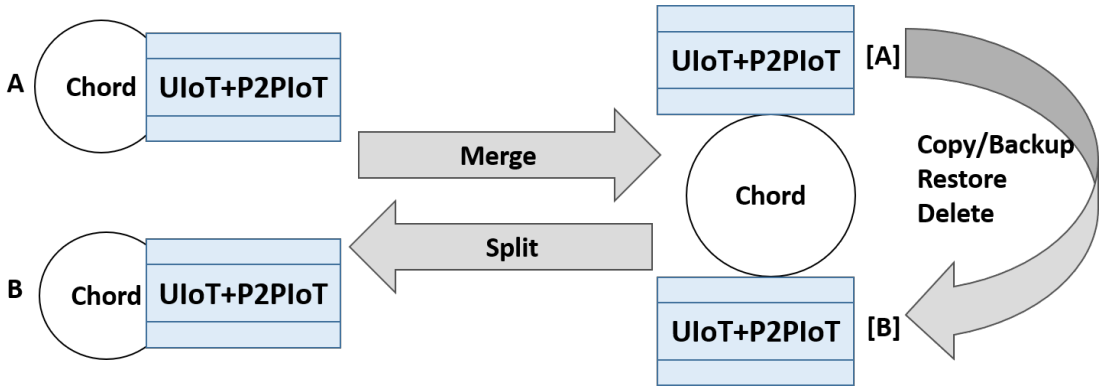


Figura 4.2: Cenário Geral de Testes de Operações do Modelo P2PIoT

Assim preservam-se as informações de dados de *smart objects* em cada instância de IoT, contornando pela operação P2P problemas tais como dificuldade de registro em diferentes redes IoT, sobrecarga de *middleware* e falhas de ponto único de mediação.

Ilustra-se desse modo no cenário de testes como a operação P2P é escalável e utilizável com outras instâncias de IoT distribuídas na internet. Reitera-se que no cenário configurado nesta tese, a rede Chord provê que os dispositivos sejam unicamente identificados pela aplicação da função de *hash* que os ordena sob a forma de um anel [4].

Assim, tendo em vista que, utilizando o protocolo Chord, as instâncias são registradas no ambiente P2P, mas também os dispositivos com suas descrições estão registrados em cada *middleware* UIoT+P2PIoT, com os respectivos endereços IP obtidos da rede de testes, têm-se o conjunto de elementos para tratar a heterogeneidade dos dispositivos e a variabilidade da informação a eles associada. Para tanto, a ideia geral dos testes é fazer a aplicação UIoT+P2PIoT se registrar no anel Chord, tendo as informações que são obtidas a partir de aplicações instaladas em diversos dispositivos móveis, onde serão capturadas as informações destes dispositivos e colocadas na base de dados do *middleware* UIoT.

Logo, uma vez o ambiente estando com suas funcionalidades já configuradas, as conexões de rede estabelecidas, os próximos passos de testes consistem em executar os serviços de interoperação para aferir os resultados.

Conforme já mencionado no início deste capítulo, para validação do modelo apresentado no capítulo 3, o protótipo UIoT+P2PIoT, operando com a configuração acima em ambiente de rede controlado, serviu para testes em dois cenários, sendo o cenário 1 de testes de interoperação P2P implementando as funcionalidades já descritas de *Backup*, *Restore* e *Delete*. Já o cenário 2 é de testes de desempenho realizados em cada transação, produzindo um comparativo entre o modelo P2P proposto e um modelo centralizado.

4.5 Cenário 1: Testes de Interoperação P2P

Para realizar os testes de serviços de interoperação da arquitetura proposta, inicialmente foi criado um ambiente controlado em uma rede local de baixa latência, instalando os componentes de *middleware*, base de dados local, serviços e a interface de usuário, conforme estrutura física apresentada na Figura 4.3 que corresponde à representação no modelo Chord conforme Figura 4.4.

A Figura 4.3 mostra o protótipo em operação na rede de testes controlada, organizada por um roteador *wi-fi* e incluindo dois componentes de hardware independentes (Raspberry Pi3), cada um com o *middleware* que recebe os dados enviados por dispositivos e realiza a persistência na respectiva base de dados, constituindo assim as duas instâncias independentes de IoT, denominadas A e B. Na Figura 4.4, as instâncias A e B são representadas com os respectivos endereços IP e correspondentes identificadores Chord.

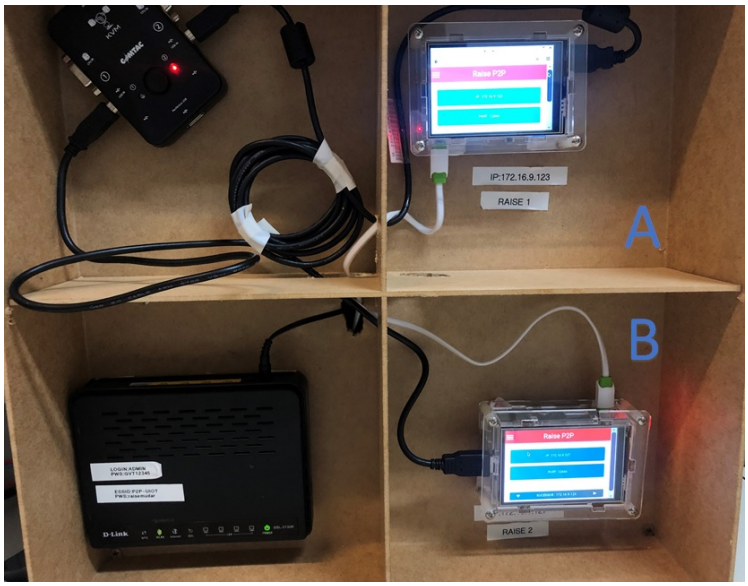


Figura 4.3: Foto do protótipo desenvolvido para testar o modelo P2PIoT

Dadas as duas instâncias de IoT A e B, representadas pelos Raspberrys com os endereços IP 172.16.9.123 e 172.16.9.127 respectivamente, conforme mostram as Figuras 4.3 e 4.4, foram realizados testes das funcionalidades de interoperação, com gerência de recursos em P2P, para as operações de copiar, agrupar e separar (*copy*, *merge*, *split*) as instâncias de IoT.

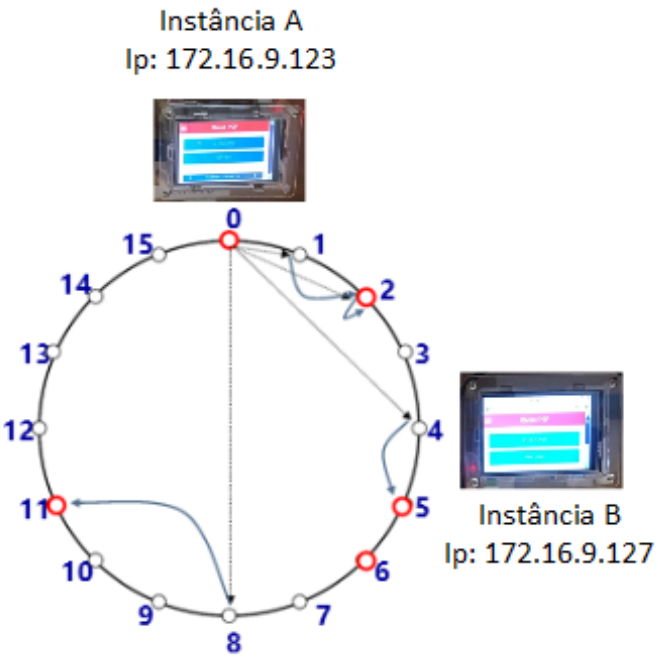


Figura 4.4: Identificação do componentes do protótipo no endereçamento IP e no Chord

Pode-se observar na Figura 4.5 a existência de 3 dispositivos na instância A, representados por 3 Raspberrys Pi do lado esquerdo. Na mesma Figura 4.5, na foto do lado direito, obtida após clicar no nome do dispositivo na foto da esquerda, mostram-se os identificadores de um dos dispositivos com suas características (Endereçamento MAC e IP, descrições de hardware, hash Chord, etc). Tais informações são as que efetivamente permitem realizar a comunicação entre as duas instâncias A e B.

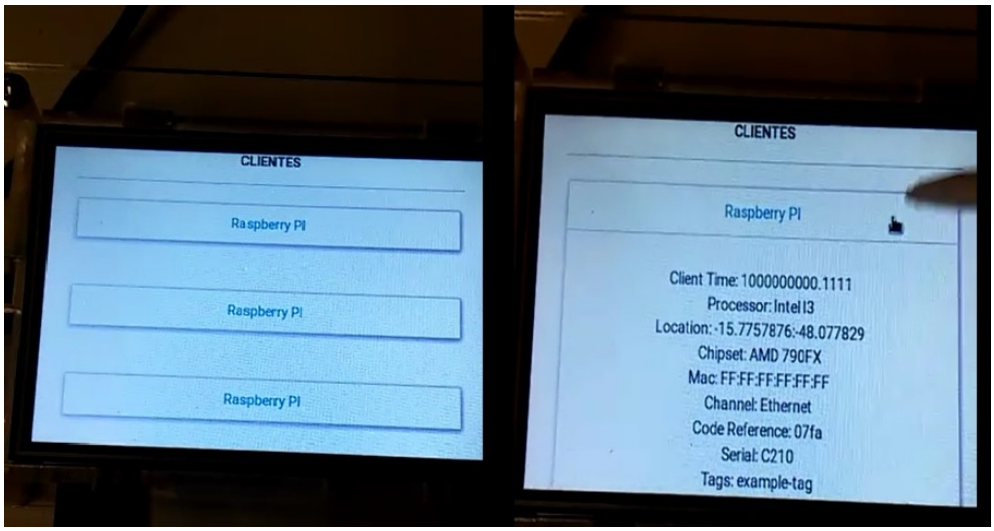


Figura 4.5: Informações de um Dispositivo da Instância A

Com essa configuração estabelecida, é possível executar os comandos referentes aos diversos serviços de interoperação do Modelo P2PIoT, conforme discutido nas próximas seções.

4.5.1 Funcionalidade de *Merge*

O *merge* trata-se de uma funcionalidade que irá buscar dados da instância A e dados da instância B, agrupando todos em uma nova instância C, conforme representado pela figura 4.6, de modo que C representa a federação, preservando as estruturas individuais de A e B.

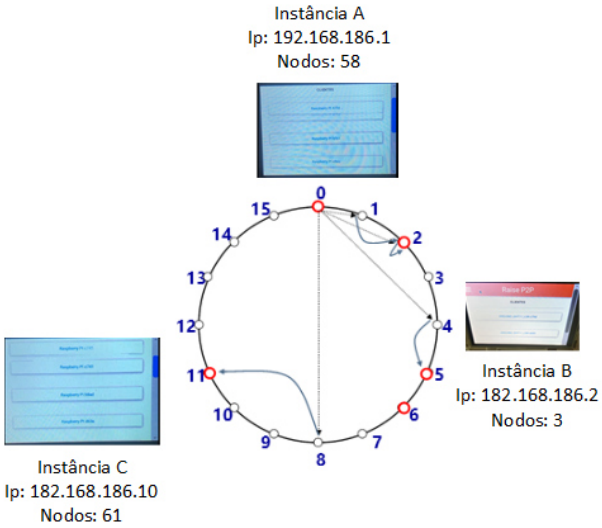


Figura 4.6: Execução da funcionalidade de *merge*

O módulo que realiza a funcionalidade de mesclagem opera nos Raspberry Pi e nestes, uma vez ativado o aplicativo de P2P, este se conecta ao *middleware* UIoT, utilizando chamadas ao protocolo HTTP e recebendo dados em formato JSON, para a construção do anel inicial e o carregamento das informações.

Para validar a funcionalidade de mesclagem, mostramos a existência de 58 dispositivos na instância A e 3 dispositivos na instância B antes de iniciar a operação. Após executar o comando de *merge*, é criada a nova instância C com 61 dispositivos, conforme pode ser verificado nas Figuras 4.7 e 4.8.

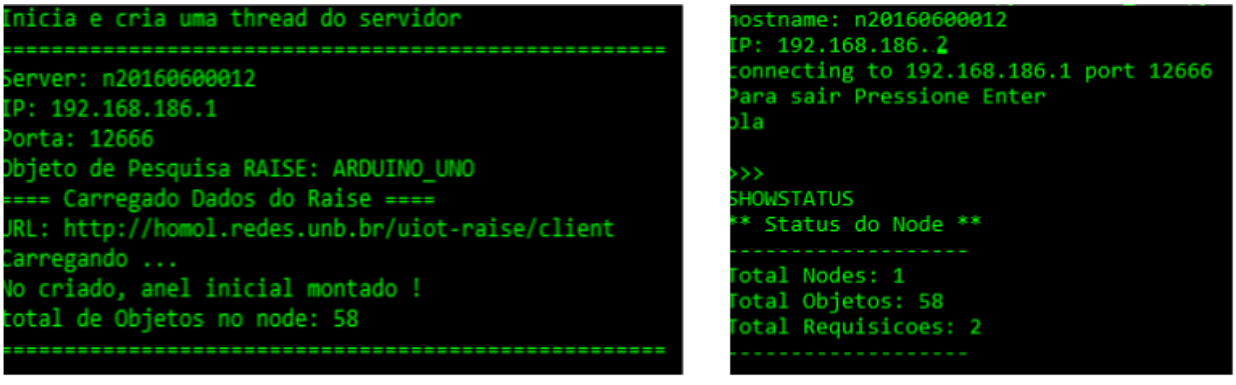


Figura 4.7: Dados das instâncias A e B antes da execução do *merge*

No intuito de visualizar os resultados da operação realizada, foi capturada a tabela de *hashes* da DHT representada na Figura 4.9 que ilustra como, uma vez criado o ambiente P2P e estabili-

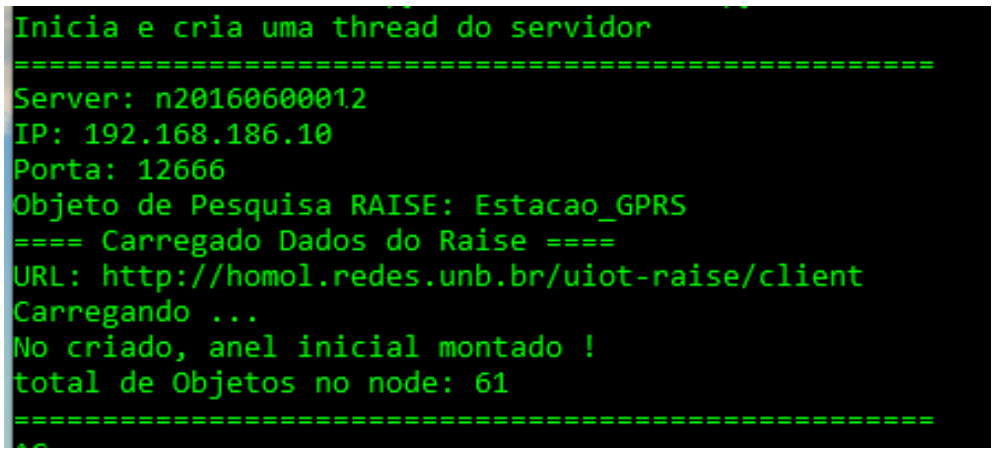


Figura 4.8: Resultado da operação da funcionalidade de *merge* na instância federada C

zadas as informações do roteamento P2P, podemos acessar o ambiente de interoperação através de diversos comandos para sabermos qual o estado em que se encontra cada nodo da rede de sobreposição.

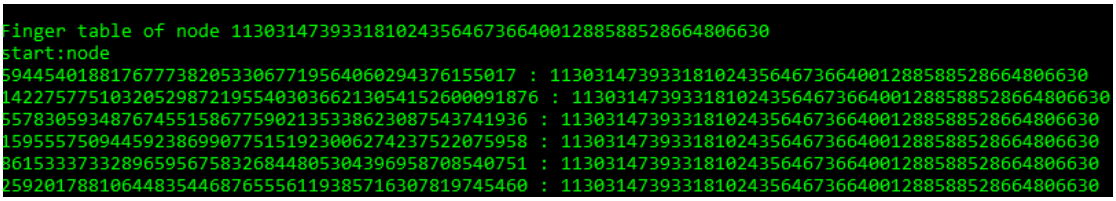


Figura 4.9: Tabela *hash* de um dos nodos Chord

Reitera-se que os dados a partir de agora são colocados em formato de DHT e que passam a fazer parte do encadeamento Chord, sendo referenciados como parte de cada nodo Chord, mas indexados globalmente na DHT. Uma vez utilizada a função de *merge* ou *join* de nodos, os dados dos diversos nodos adicionados irão aparecer como adicionados tanto nas tabelas informacionais de cada *middleware* como também nos dados de nodos Chord.

A função de *merge* associa os dados do nodo que inicia o pedido de mesclagem com a sua respectiva tabela informacional, mantendo a estrutura local de associação e adicionando no Chord a linha informando qual o próximo nodo no roteamento do anel Chord. Como cada nodo adiciona uma linha informando o próximo nodo e assim sucessivamente, cria-se o anel que se fecha na tabela do nodo que inicializou o processo.

Com o anel P2P estabelecido, é possível acessar cada um dos dispositivos de cada uma das instâncias federadas, realizar as conexões entre as instâncias e executar as demais funcionalidades propostas na arquitetura P2PIoT.

Na Figura 4.10, por exemplo, é mostrado um determinado dispositivo já com o menu das possíveis funcionalidades de *Copy/Backup* e de *Restore*, de forma que podemos fazer as operações para esse dispositivo selecionado, a começar pela obtenção de dados desse dispositivo conforme Figura 4.11. As demais funcionalidades são discutidas a seguir.

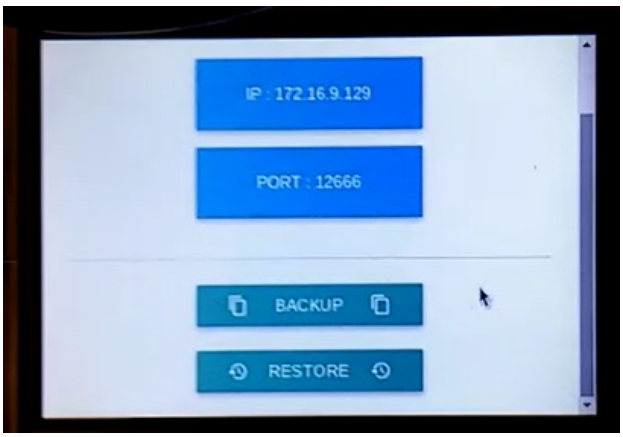


Figura 4.10: Dados e menu de operações para a instância federada de IoT

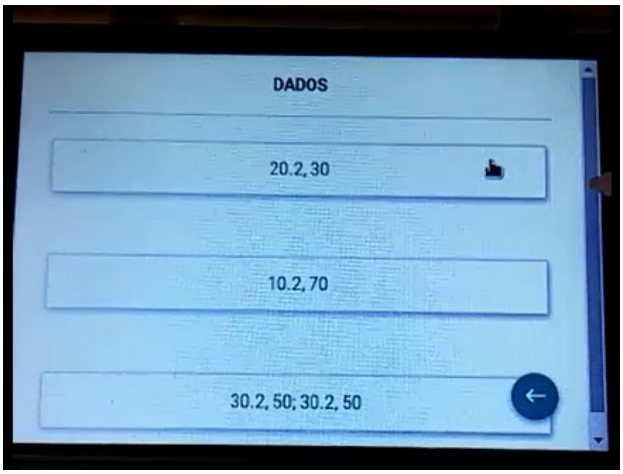


Figura 4.11: Apresentação dos dados de um dispositivo na instância federada de IoT

4.5.2 Funcionalidade de *Copy/Backup*

A funcionalidade de *Copy/Backup* trata-se de um módulo que copiar a base informacional de dispositivos (Raspberry) de uma instância para outra, conforme comando em execução mostrado na figura 4.12, caso em que o dispositivo é copiado da instância B (172.16.9.127) para a instância A (172.16.9.123). Feita a copia dos dados do dispositivo, este pode ser removido da instância e eventualmente ser reinstalado nesta ou em outra instância, haja vista a possibilidade de restaurar as informações copiadas conforme descrito em seguida.

4.5.3 Funcionalidade de *Restore*

O funcionalidade de *Restore* trata-se de um módulo que irá realizar a copia a base informacional referente ao um dispositivo de uma instância, base informacional essa que possivelmente tenha sido apagada ou que seja instalada um novo dispositivo em outra instância. No ambiente controlado com dispositivos Raspberry, no caso do experimento, foi utilizado como origem dos dados o nodo A, IP 172.16.9.123, que restaura as informações para o nodo B, IP 172.16.19.127, conforme mostra a figura 4.13.

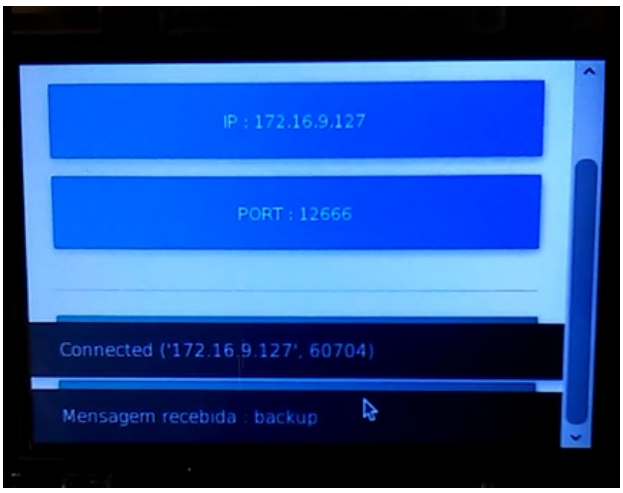


Figura 4.12: Operação de *Backup* no Modelo P2PIoT

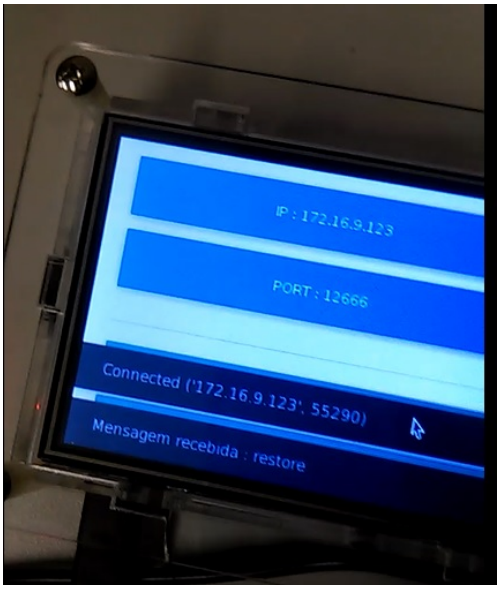


Figura 4.13: Operação de *Restore* no Modelo P2PIoT

4.5.4 Funcionalidade de *Delete*

A função *delete* é uma das funções mais simples utilizadas no modelo, entretanto de extrema importância, pois se refere a um módulo responsável por apagar uma instância de IoT, como mostra a figura 4.14. Uma vez a instância apagada, a mesma só poderá ser recuperada através de funcionalidade de *restore* a partir de uma cópia previamente realizada. No cenário de testes da tese, realizamos a funcionalidade de copiar uma base informacional de uma instância A para uma instância B e, em seguida, realizamos a função de *delete* na instância, obtendo o resultado mostrado na Figura 4.15.

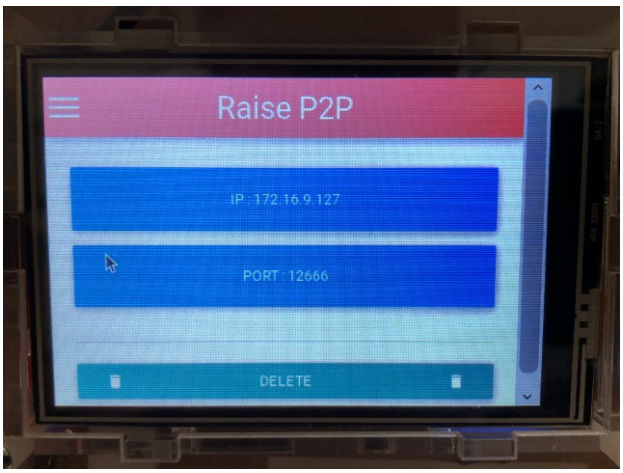


Figura 4.14: Tela da função de Delete

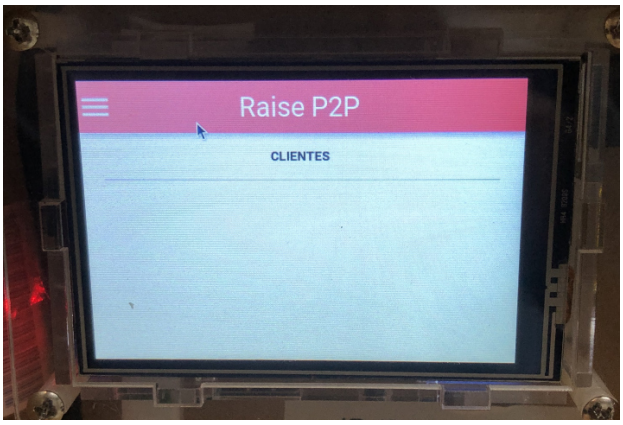


Figura 4.15: Instância após a execução do comando delete

4.5.5 Funcionalidade de *Split*

Esta operação é realizada no nodo C para remover os dados de um nodo B que foram recebidos de um *merge* anterior de A com B. Caso o nodo B ainda esteja ativo então o *split* apenas irá remover os dados de B do nodo C. Caso o nodo B não mais exista, então os dados do antigo nodo B serão passados para um novo nodo sucessor no anel de Chord, e dados de B serão removidos de C.

4.6 Cenário 2: Testes de Desempenho e Resultados Comparativos

4.6.1 Teste de Desempenho

Tendo em vista que este trabalho tem como objetivo uma proposta de um modelo de interoperabilidade de instâncias IoT em um modelo P2P e que para validar tal proposta é importante obter suas características de desempenho, foi proposto o Cenário 2 para testes de desempenho que

permitam obter resultados comparando a arquitetura P2PIoT com uma arquitetura centralizada de interoperação.

Os testes comparativos das duas arquiteturas visam provar que a arquitetura em P2P é viável para a interoperabilidade de instâncias IoT, não apenas por prover as mesmas funcionalidades, mas por apresentar desempenho no mínimo similar ao de arquiteturas centralizadas.

Para tanto, além de testes com o *middleware* UIoT+P2PIoT, foram feitos testes com uma versão do *middleware* UIoT implementada em nuvem sem o serviço P2PIoT, servindo os resultados destes últimos como referência para comparar com as medições de desempenho do Modelo P2PIoT.

Para os testes de desempenho com o *middleware* UIoT+P2PIoT, foi configurado o cenário da Figura 4.16, onde duas instâncias UIoT+P2PIoT são colocadas em uma rede local, com uma delas (*Tracker*) sendo responsável pela inicialização do anel Chord. Entre tais instâncias, são executadas baterias de teste com os serviços de interoperação *Copy* e *Restore* disponíveis no P2PIoT. As baterias de testes são repetitivas e os resultados médios de 10 execuções são calculados para efeito da análise.

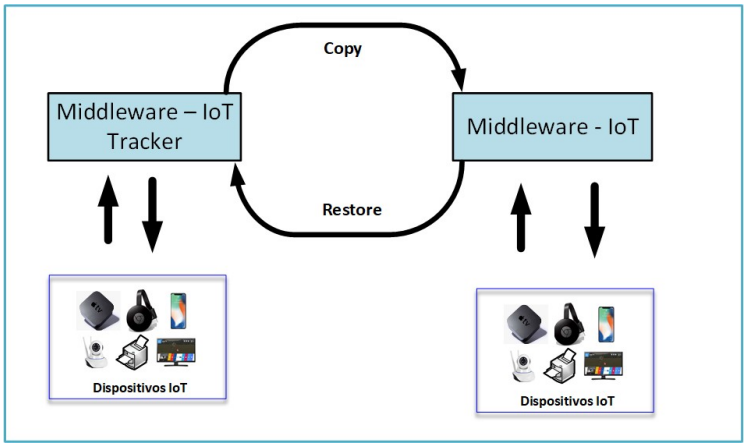


Figura 4.16: Cenário de interoperação em ambiente local em Modelo P2PIoT

As baterias de teste foram repetidas para os serviços de interoperação *Copy* e *Restore* referentes a 1, 10, 100 dispositivos, dando resultados para estudo comparativo quanto a volumetria e tempo de resposta (busca e gravação), quanto a consumo de memória. Os dados médios de volumetria para 10 execuções são apresentados na Figura 4.17, enquanto os demais resultados são apresentados mais adiante já em comparação com um modo de interoperação centralizado.

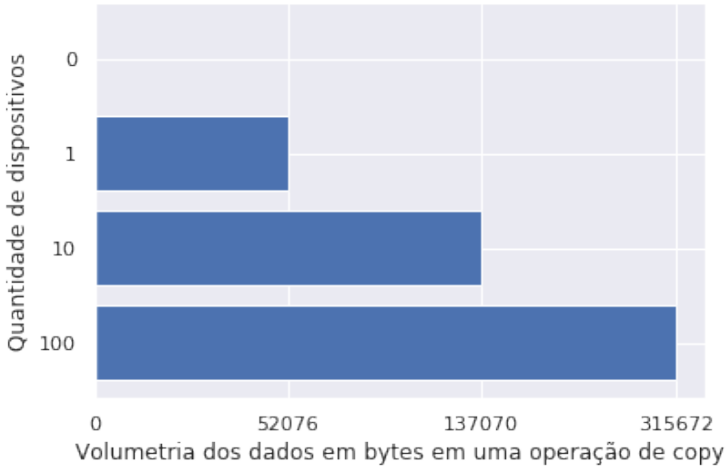


Figura 4.17: Resultados de volumetria dos testes realizados no Cenário 2 com o UIoT+P2PIoT

Para os testes de desempenho com o *middleware* UIoT em nuvem, ou seja sem P2PIoT, foi configurado o cenário da Figura 4.18 no qual a comunicação dos *middlewares* se passa em uma nuvem que se encontra fora do Laboratório UIoT, mais especificamente em *datacenter* da Rede Nacional de Pesquisa - RNP, rede que provê serviços de acesso à Internet para a UnB, de maneira que configura uma situação em se pode considerar o tráfego que foi gerado para a nuvem, utilizando a Internet como meio de comunicação, nos testes de funcionalidades de interoperação tais como previstas na arquitetura proposta nesta tese.

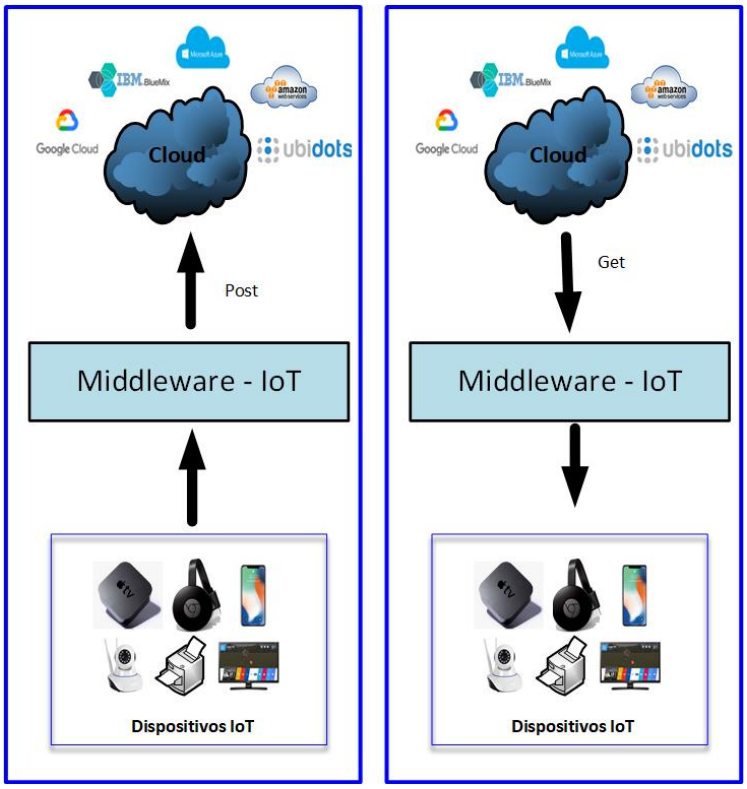


Figura 4.18: Modelo de interoperação usando o *middleware* UIoT em ambiente de nuvem

Com o cenário da Figura 4.18, foram também realizados os mesmos testes de desempenho

com 1, 10 e 100 dispositivos já feitos para o UIoT+P2PIoT. Os dados médios de volumetria para 10 execuções foram obtidos. Para efeito de ilustração, os resultados da operação de cópia de 1 dispositivo são apresentados na Tabela 4.1. Tais valores são interessantes como referência de comparação entre os dois modelos de interoperação. Já os demais resultados das baterias de testes são apresentados mais adiante, em comparação com um modo de interoperação P2P.

Tabela 4.1: Resultados do teste do *UIoT* utilizando nuvem para operação de cópia de 1 dispositivo

Métricas	Resultado
Memória Consumida	107 MB
Tempo de Gravação	1,5 minuto
Tempo de Restore	1,27 minuto

4.6.2 Comparativo de Resultados

Conforme já apresentado, para efeito de comparação, foram realizados os mesmos testes e das mesmas operações *copy* e *restore*, tanto com o protótipo do modelo P2PIoT quanto com a versão do UIoT operando em nuvem. Tais testes correspondem à cópia de dispositivos do grupo [A] para o grupo [B] da instância federada mostrada no cenário da Figura 4.2.

Apresentam-se a seguir resultados médios de 10 execuções de baterias de cópias do mesmo grupo de dispositivo de uma instância de IoT para a outra, para as quantidades de 1, 10, 100 e 1000 dispositivos. Não dispondo de ambiente com essa quantidade de dispositivos reais, as operações são na verdade a partir das abstrações de dispositivos definidas virtualmente na configuração adotada.

A Figura 4.19 apresenta os resultados para a análise de consumo de memória comparativa entre o ambiente UIoT em nuvem (linha azul no gráfico) e o ambiente UIoT+P2PIoT (linha vermelha no gráfico). Verifica-se que o consumo de memória no UIoT em nuvem foi maior que no ambiente UIoT+P2PIoT, o que se deve a uma série de fatores, dentre eles os processos de autenticação e concorrência na organização das interações com a nuvem.

Pode-se observar que, em todos os casos, o ambiente de nuvem requer mais memória, com pequena tendência de aumentar ainda mais esse requisito com o aumento no número de dispositivos envolvidos no serviço de interoperação.

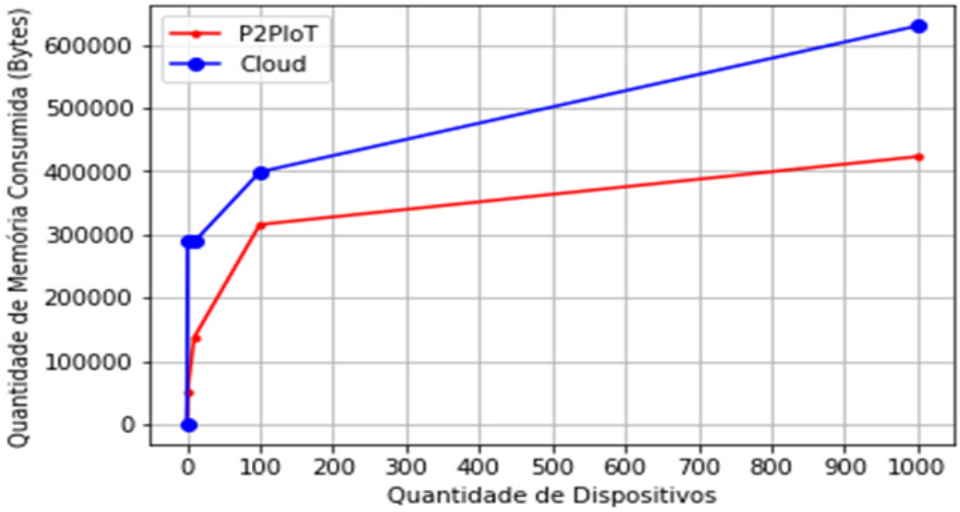


Figura 4.19: Resultados de consumo de memória nos testes de interoperação comparados entre o Modelo P2PIoT e o modelo centralizado

Já na figura 4.20, apresentam-se os resultados de desempenho relativos a tempo utilizado para os serviços de interoperação, notando a utilização de escala logarítmica no eixo horizontal. Observa-se que o ambiente UIoT em nuvem apresenta maior tempo de resposta que o UIoT+P2PIoT, por estar utilizado diversas sub-redes heterogêneas e roteamento para acessar a nuvem, enquanto o no ambiente P2PIoT as conexões se tornam mais diretas e consequentemente o tempo de resposta é menor.

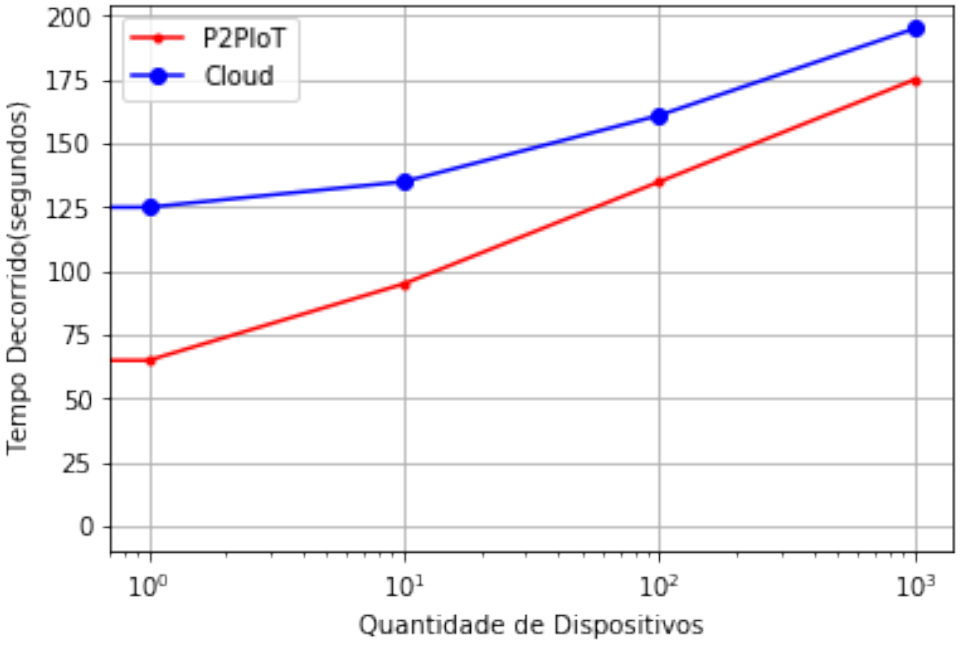


Figura 4.20: Resultados de tempo de resposta nos testes de interoperação comparados entre o Modelo P2PIoT e o modelo centralizado

Conforme mostram as figuras, o desempenho do modelo utilizando a arquitetura P2PIoT sobre

o Chord consumiu menos memória e apresentou menor tempo de execução em comparação ao modelo de interoperação intermediado por uma entidade central operando em nuvem. Observa-se que em ambos os casos, a quantidade de memória consumida está relativamente ao alcance em configurações realistas de instâncias de redes IoT com número de até 1000 dispositivos. Já no que se refere ao tempo, a execução de serviços relativamente complexos de interoperação implica em gasto de tempo que deve ser considerado em ambientes dinâmicos nos quais os dispositivos de IoT possam eventualmente ser desligados ou eventualmente estejam com mobilidade, pois tal tempo de resposta será fundamental para manter a coerência da informação de estado dos dispositivos gerenciada pelo *middleware* de IoT.

Capítulo 5

Conclusão

Nesta tese, partimos da definição do conceito de **Instância de IoT** e, com apoio de estudo da literatura (Capítulo 2), caracterizamos a necessidade de interoperação entre diferentes instâncias de IoT, pressupondo que as interações internas a cada instância sejam tarefa do *middelware* de controle local que provê identificação e fluxos de informação para os dispositivos e as aplicações dentro de cada instância. De fato, sendo responsável pela integração e comunicação entre componentes heterogêneos, o *middleware* IoT deve fornecer um conjunto de abstrações de programação, incluindo descoberta de dispositivos, acesso a serviços e tarefas de administração.

O que se verifica na evolução recente da IoT é a existência de plataformas de hardware e sistemas operacionais potencialmente diferentes, assim como *middleware*, protocolos de comunicação, modelos de dados, tipos de aplicativos e políticas de segurança, entre outros componentes. Assim, em função de necessidades locais do setor ou do ambiente de aplicação, ocorre a criação ou instalação de múltiplas redes IoT setorizadas, cada uma com sua estrutura própria de dispositivos, sistemas operacionais, bases de dados, *middleware* e aplicações. Cabe então uma definição mais realista que se refere ao conceito de várias instâncias de redes IoT em execução paralela e simultânea. Tais instâncias independentes se configuram também em razão de separação espacial ou de domínio administrativo, mesmo quando se considera a utilização de um conjunto de hardware e software homogêneo.

O estudo da literatura indica que essas características juntas acarretam os seguintes desafios de pesquisa: heterogeneidade dos sistemas de IoT, baixa interoperabilidade entre esses sistemas, restrições de recursos dos dispositivos de IoT, bem como vulnerabilidades de privacidade e segurança.

Considerando tal situação, existe a possibilidade, e mesmo a necessidade, de interações de uma instância de IoT para outra, seja entre componentes específicos, seja na forma de operações envolvendo a integralidade dos elementos de duas ou mais dessas instâncias de IoT. Designamos como **interoperação entre instâncias de IoT**, o conjunto de procedimentos, protocolos e operações necessários para dar suporte a tais interações de uma instância de IoT para outra.

Especificamente, a propriedade de interoperabilidade é definida como a capacidade de inte-

ragir com sistemas físicos e trocar informações entre sistemas IoT. Por consequente, definimos o conceito de **serviços de interoperação IoT** como sendo funcionalidades que o *middleware* controlador de uma instância de IoT pode utilizar para interagir com outro *middleware* que opera noutra instância de IoT potencialmente diferente da primeira, em termos de estrutura e componentes e setor/ambiente de aplicação.

Em resposta à necessidade de interoperação entre instâncias de IoT, concebemos um **Modelo de Interoperação, denominado P2PIoT**, fundado na hipótese (Capítulo 1) de que uma solução entre sistemas pares, *peer-to-peer* – P2P, permitiria a criação dos necessários **serviços de interoperação P2P para IoT** e com características de transparência, desempenho e escalabilidade adequados, em particular na comparação com uma solução de interoperação sob controle de um sistema centralizador.

Assim, respondemos conceitualmente à hipótese adotada, com a proposição de que a interoperabilidade de IoT pode ser alcançada por meio de uma camada de software em cima de uma rede de sobreposição P2P com acesso uniforme em diferentes sistemas de IoT.

Ademais, esta tese se desenvolve além da conceituação básica, com a especificação de serviços de interoperabilidade e sua realização sobre um sistema de busca P2P. **O modelo P2PIoT objetiva então a interoperação de diferentes instâncias de IoT usando serviços de interoperação de IoT sobrepostos a um serviço de busca P2P**.

Para tanto, os serviços de busca e interoperação são integrados ao *middleware* em cada instância de IoT para fornecer operações destinadas a federar as instâncias heterogêneas de IoT em cenários fixos e móveis. O modelo proposto suporta serviços para mesclar e dividir instâncias de IoT, bem como para mover, replicar e excluir dados da IoT, ou de suas entidades de software e abstrações de dispositivos.

Desse modo, esta tese, no que se refere especificamente à segurança, objetiva a propriedade de disponibilidade de serviços e de dados, nos sentido de que lógica de funcionamento do sistema deve ser integrada a políticas de execução da segurança, nos seus demais aspectos que não são objeto desta tese. Em consequência, esta tese é dedicada à investigação de uma proposta de funcionalidade para a interoperação de IoT, de modo a permitir o teste da efetividade e do desempenho dessa funcionalidade, considerando para outros estudos o aspecto geral da segurança, mas pressupondo que medidas de segurança distribuídas de IoT estejam ou possam ser estabelecidas em cada instância de IoT. Vale observar que o modelo de interoperação proposto nesta tese, sendo da classe P2P, adequa-se ao acoplamento de extensões no sentido de integrar o consenso e o registro das transações de modo distribuído, de modo análogo ao protocolo *blockchain* e outros algoritmos de consenso.

O Modelo P2PIoT, após ter sua estrutura e funcionamento especificados e discutidos (Capítulo 3), foi desenvolvido na forma de um protótipo usado na validação (Capítulo 4) cujos resultados, obtidos a partir da avaliação em 3 cenários de testes em laboratório controlado, indicam que a hipótese adotada na tese teve sucesso na concepção e na experimentação. Assim, a proposta P2PIoT é validada pelo teste de um protótipo desenvolvido e pela discussão de suas funcionalidades, es-

calabilidade e desempenho, notando que esta solução P2P naturalmente se adequa à computação na borda e nos dispositivos, mostrando-se quanto a tais aspectos mais promissora que as soluções centralizadas e organizadas somente com processamento em nuvem.

Para efeito de conclusão geral, esta tese resulta em **duas contribuições quanto à interoperação em IoT**, haja vista que o modelo proposto, P2PIoT, contempla um **módulo para serviços de interoperabilidade** acoplado a um **módulo de interface de programação para serviços de busca P2P**. Outras **contribuições complementares** vêm do processo de validação, que inclui o desenvolvimento de um **protótipo para avaliar os serviços de interoperação**, bem como a discussão de **resultados de escalabilidade e desempenho na presença de um número variável de dispositivos**, e de forma comparativa a alternativa solução centralizada.

5.1 Trabalhos Futuros

Conforme visto, o Modelo P2PIoT proposto define funcionalidades de interoperação para instâncias de IoT e o correspondente protótipo desenvolvido valida as características desejadas para as funcionalidades propostas.

Mas, considerando que a validação foi realizada em ambiente de laboratório controlado, é importante que testes mais amplos em cenários menos controlados, ou seja com objetos de IoT presentes na Internet aberta, sejam feitos para obter uma melhor caracterização da proposta diante de tráfego e da mobilidade realista de instâncias heterogêneas de IoT.

Por outro lado, é preciso avançar na análise com relação aos aspectos de confidencialidade e integridade, vistos globalmente em uma política de segurança aplicável a uma federação de instâncias de IoT heterogêneas. Problemas específicos de segurança, a parte aqueles relativos à disponibilidade de serviços de interoperação, não foram tratados nesta tese. Dessa forma são sugeridos, como proposta de trabalhos futuros, os itens:

- Propor um modelo de segurança para troca de informações em federações de instâncias de IoT;
- Aprimorar o modelo proposto, propondo uma abstração do serviço de busca P2P utilizado, de modo a ter a opção de usar outros serviços P2P, como *blockchain*, caso o modelo de segurança requisite a utilização de consenso distribuído em federações de IoT;
- Estudar adaptações do modelo proposto para atender a questões de desempenho e plasticidade com relação à implementação de mecanismos de segurança.

Feitas tais considerações, a exploração de trabalhos futuros significa, com certeza, descobrir necessidades de interoperação não contempladas nesta tese. IoT é ainda um campo fértil para inovações e seria ilusório assumir que apenas um trabalho de pesquisa possa cobrir todo o assunto. Nesse sentido, esta tese pode ser vista como uma contribuição no contexto de um campo de pesquisa em aberto e como uma baliza para um novo ciclo de pesquisa e inovação.

Referências Bibliográficas

[1] ALMEIDA, M. P. D. et al. New dos defense method based on strong designated verifier signatures. *Sensors*, v. 18, n. 9, 2018. ISSN 1424-8220. Disponível em: <http://dx.doi.org/10.3390/s18092813>.

[2] SILVA, C. C. d. M. et al. Proposta de auto-registro de serviços pelos dispositivos em ambientes de iot. *34º Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2016.

[3] HUACARPUMA, R. C. Proposição de um modelo e sistema de gerenciamento de dados distribuídos para internet das coisas–gddiot. 2017. Disponível em: <http://repositorio.unb.br/handle/10482/31146>.

[4] STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, ACM, v. 31, n. 4, p. 149–160, 2001. ISSN 0146-4833. Disponível em: <http://dx.doi.org/10.1145/383059.383071>.

[5] GHODSI, A. *Distributed k-ary system: Algorithms for distributed hash tables*. Tese (Doutorado) — KTH-Royal Institute of Technology, 2006.

[6] AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. In: . IEEE, 2015. v. 17, n. 4, p. 2347–2376. ISBN 1553-877X. Disponível em: <http://dx.doi.org/10.1109/COMST.2015.2444095>.

[7] PARTNERS oneM2M. Ts-0001 functional architecture (v2.18.1). 2018. Disponível em: <http://www.onem2m.org/component/rsfiles/>.

[8] DAI, H.-N.; ZHENG, Z.; ZHANG, Y. Blockchain for internet of things: A survey. *arXiv preprint arXiv:1906.00245*, 2019. Disponível em: <http://dx.doi.org/10.1109/JIOT.2019.2920987>.

[9] RIZZARDI, A. et al. Aups: An open source authenticated publish/subscribe system for the internet of things. *Information Systems*, Elsevier, 2016. ISSN 0306-4379. Disponível em: <http://dx.doi.org/10.1016/j.is.2016.05.004>.

[10] FERREIRA, H.; JUNIOR, R. de S. Security analysis of a proposed internet of things middleware. *Cluster Computing*, v. 20, n. 1, p. 651–660, 2017. Disponível em: <http://dx.doi.org/10.1007/s10586-017-0729-3>.

[11] FERREIRA, H. G. C. et al. Proposal of a secure, deployable and transparent middleware for internet of things. In: *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*. [s.n.], 2014. p. 1–4. ISSN 2166-0727. Disponível em: <http://dx.doi.org/10.1109/CISTI.2014.6877069>.

[12] DUTRA, B. V. et al. HIDS by signature for embedded devices in iot networks. In: *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019)*. [S.l.: s.n.], 2019. v. 1, p. 53–61. ISBN 978-84-09-12121-2.

[13] FARRIS, I. et al. Federated IoT services leveraging 5g technologies at the edge. *Ad Hoc Networks*, v. 68, p. 58 – 69, 2018. ISSN 1570-8705. Advances in Wireless Communication and Networking for Cooperating Autonomous Systems. Disponível em: <https://doi.org/10.1016/j.adhoc.2017.09.002>.

[14] ZANATTA, M. da R. et al. Tensor-based time-delay estimation for second and third generation global positioning system. *Digital Signal Processing*, Elsevier, 2019. Disponível em: <http://dx.doi.org/10.1016/j.dsp.2019.04.003>.

[15] TORRES, J. A. S. et al. Melhoria da precisão dos indicadores na governança digital de serviços públicos à vista da análise de bases de dados de empregabilidade. *Inclusão Social*, v. 12, n. 2, 2019.

[16] SILVA, D. A. da et al. Produção de indicadores de empregabilidade com base em técnicas de mineração de big data e business intelligence. *Inclusão Social*, v. 12, n. 2, 2019.

[17] PRACIANO, B. J. G. et al. Spatio-temporal trend analysis of the brazilian elections based on twitter data. In: IEEE. *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2018. p. 1355–1360. Disponível em: <http://dx.doi.org/10.1109/ICDMW.2018.00192>.

[18] MARTINS, L. et al. Design and evaluation of a semantic gateway prototype for iot networks. In: ACM. *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. 2017. p. PS–7. Disponível em: <http://dx.doi.org/10.1145/3147234.3148091>.

[19] FILHO, F. L. de C. et al. Gerenciamento de serviços iot com gateway semântico. In: IADIS PRESS. *Atas das Conferências Ibero-Americanas WWW/Internet e Computação Aplicada*. [S.l.], 2017. p. 199–206.

[20] PEFFERS, K. et al. A design science research methodology for information systems research. *Journal of management information systems*, Taylor & Francis, v. 24, n. 3, p. 45–77, 2007. ISSN 0742-1222. Disponível em: <http://dx.doi.org/10.2753/MIS0742-1222240302>.

[21] ASHTON, K. That 'internet of things' thing. *RFiD Journal*, v. 22, p. 97–114, 01 2009.

[22] BAUER, M.; WALEWSKI, J. W. The iot architectural reference model as enabler. In: ____. *Enabling Things to Talk*. Springer, Berlin, Heidelberg, 2013. p. 17–25. ISBN 978-3-642-40402-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-40403-0$_3$>.

[23] Zhu, F.; Mutka, M. W.; Ni, L. M. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, v. 4, n. 4, p. 81–90, Oct 2005. ISSN 1536-1268. Disponível em: <http://dx.doi.org/10.1109/MPRV.2005.87>.

[24] GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2013.01.010>.

[25] HAMRAZ, S. H. Internet of things. *Computer Science*, 2013. Disponível em: <http://dx.doi.org/10.13140/RG.2.2.22522.90563>.

[26] ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. ISSN 1389-1286. Disponível em: <https://doi.org/10.1016/j.comnet.2010.05.010>.

[27] TOMA, I.; SIMPERL, E.; HENCH, G. A joint roadmap for semantic technologies and the internet of things. In: *Proceedings of the Third STI Roadmapping Workshop, Crete, Greece*. [S.l.: s.n.], 2009. v. 1, p. 140–53.

[28] ZHANG, D. et al. Survey on context-awareness in ubiquitous media. *Multimedia tools and applications*, Springer, v. 67, n. 1, p. 179–211, 2013. Disponível em: <http://dx.doi.org/10.1007/s11042-011-0940-9>.

[29] MATTERN, F.; FLOERKEMEIER, C. From the internet of computers to the internet of things. In: *From active data management to event-based systems and more*. Springer, 2010. p. 242–259. ISBN 1432-122X. Disponível em: <http://dx.doi.org/10.1007/978-3-642-17226-7_15>.

[30] COOPER, J.; JAMES, A. Challenges for database management in the internet of things. *IETE Technical Review*, Taylor Francis, v. 26, n. 5, p. 320–329, 2009. Disponível em: <http://dx.doi.org/10.4103/0256-4602.55275>.

[31] ABU-ELKHEIR, M.; HAYAJNEH, M.; ALI, N. A. Data management for the internet of things: Design primitives and solution. In: ____. [s.n.], 2013. v. 13, n. 11, p. 15582–15612. Disponível em: <http://dx.doi.org/10.3390/s131115582>.

[32] Cecchinel, C. et al. An architecture to support the collection of big data in the internet of things. In: *2014 IEEE World Congress on Services*. [s.n.], 2014. p. 442–449. ISSN 2378-3818. Disponível em: <http://dx.doi.org/10.1109/SERVICES.2014.83>.

[33] VONGSINGTHONG, S.; SMANCHAT, S. A review of data management in internet of things. *Asia-Pacific Journal of Science and Technology*, v. 20, n. 2, p. 215–240, 2015. Disponível em: <http://dx.doi.org/10.14456/kkurj.2015.18>.

[34] RAFIEI, D.; MENDELZON, A. Similarity-based queries for time series data. In: . New York, NY, USA: ACM, 1997. v. 26, n. 2, p. 13–25. ISSN 0163-5808. Disponível em: <http://dx.doi.org/10.1145/253262.253264>.

[35] BUENO, R. D. L. da S. *Econometria de séries temporais*. [S.l.]: Cengage Learning, 2008. ISBN 9788522111572.

[36] HUACARPUMA, R. C. et al. Distributed data service for data management in internet of things middleware. *Sensors*, v. 17, n. 5, 2017. ISSN 1424-8220. Disponível em: <http://dx.doi.org/10.3390/s17050977>.

[37] BORGIA, E. The internet of things vision: Key features, applications and open issues. In: . Elsevier, 2014. v. 54, p. 1–31. ISBN 0140-3664. Disponível em: <http://dx.doi.org/10.1016/j.comcom.2014.09.008>.

[38] FERSI, G. Middleware for internet of things: A study. In: IEEE. *2015 International Conference on Distributed Computing in Sensor Systems*. 2015. p. 230–235. ISSN 2325-2936. Disponível em: <http://dx.doi.org/10.1109/DCOSS.2015.43>.

[39] CHAQFEH, M. A.; MOHAMED, N. Challenges in middleware solutions for the internet of things. In: IEEE. *2012 international conference on collaboration technologies and systems (CTS)*. 2012. p. 21–26. ISBN 978-1-4673-1381-0. Disponível em: <http://dx.doi.org/10.1109/CTS.2012.6261022>.

[40] HANSMANN, U. et al. *Pervasive computing handbook*. Springer Science & Business Media, 2013. ISBN 978-3-662-04318-9. Disponível em: <http://dx.doi.org/10.1007/978-3-662-04318-9>.

[41] PERERA, C. et al. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 16, n. 1, p. 414–454, 2013. Disponível em: <http://dx.doi.org/10.1109/SURV.2013.042313.00197>.

[42] BRAY, T. The javascript object notation (json) data interchange format. 2014. Disponível em: <http://dx.doi.org/10.17487/RFC7158>.

[43] NURSEITOV, N. et al. Comparison of json and xml data interchange formats: a case study. *Caine*, v. 9, p. 157–162, 2009.

[44] RAZZAQUE, M. A. et al. Middleware for internet of things: a survey. *IEEE Internet of things journal*, IEEE, v. 3, n. 1, p. 70–95, 2015. Disponível em: <http://dx.doi.org/10.1109/JIOT.2015.2498900>.

[45] MAMEI, M.; ZAMBONELLI, F. *Field-based coordination for pervasive multiagent systems*. Springer Science & Business Media, 2006. ISBN 978-3-540-27969-3. Disponível em: <http://dx.doi.org/10.1007/3-540-27969-5>.

[46] COSTA, P. et al. Teenylime: transiently shared tuple space middleware for wireless sensor networks. In: ACM. *Proceedings of the international workshop on Middleware for sensor networks*. 2006. p. 43–48. ISBN 1-59593-424-3. Disponível em: <http://dx.doi.org/10.1145/1176866.1176874>.

[47] AZZARA, A. et al. Middleware solutions in wsn: The iot oriented approach in the icsi project. In: IEEE. *2013 21st International Conference on Software, Telecommunications and Computer Networks-(SoftCOM 2013)*. 2013. p. 1–6. ISBN 978-953-290-040-8. Disponível em: <http://dx.doi.org/10.1109/SoftCOM.2013.6671886>.

[48] MORENO, R. A. Interoperabilidade de sistemas de informação em saúde. *Journal of Health Informatics*, v. 8, n. 3, 2016.

[49] MUCHERONI, M. L.; SILVA, J. F. M. da. A interoperabilidade dos sistemas de informação sob o enfoque da análise sintática e semântica de dados na web. *PontodeAcesso*, v. 5, n. 1, p. 3–18, 2011. Disponível em: <http://dx.doi.org/10.9771/1981-6766rpa.v5i1.3622>.

[50] GIMÉNEZ, J. et al. Arsenic sorption onto natural hematite, magnetite, and goethite. *Journal of hazardous materials*, Elsevier, v. 141, n. 3, p. 575–580, 2007. Disponível em: <http://dx.doi.org/10.1016/j.jhazmat.2006.07.020>.

[51] COADIC, Y.-F. L. *Usages et usagers de l'information*. [S.l.]: Armand Colin, 2004.

[52] FARINELLI, F.; MELO, S.; ALMEIDA, M. B. O papel das ontologias na interoperabilidade de sistemas de informação: reflexões na esfera governamental. 2014.

[53] MARCONDES, C. H.; SAYÃO, L. F. Integração e interoperabilidade no acesso a recursos informacionais eletrônicos em c&t: a proposta da biblioteca digital brasileira. *Ciência da Informação*, SciELO Brasil, v. 30, n. 3, p. 24–33, 2001. Disponível em: <http://dx.doi.org/10.1590/S0100-19652001000300004>.

[54] SILVEIRA, L. M. C. d.; RIBEIRO, V. M. B. Grupo de adesão ao tratamento: espaço de"ensinagem"para profissionais de saúde e pacientes. *Interface-Comunicação, saúde, educação*, SciELO Public Health, v. 9, p. 91–104, 2005. Disponível em: <http://dx.doi.org/10.1590/S1414-32832005000100008>.

[55] FISCHELL, R. E.; FISCHELL, D. R.; UPTON, A. R. *Data recording methods for an implantable device*. [S.l.]: Google Patents, mar. 19 2002. US Patent 6,360,122.

[56] ANDRADE, J.; LARA, M. L. G. de. Interoperabilidade e mapeamentos entre sistemas de organização do conhecimento: Bioportal of national center for biomedical ontology-ncbo. *Revista de Saúde Digital e Tecnologias Educacionais*, v. 3, p. 43–61, 2018. Disponível em: <http://dx.doi.org/10.1093/nar/gkr469>.

[57] PATEL, P.; CASSOU, D. Enabling high-level application development for the internet of things. *Journal of Systems and Software*, v. 103, p. 62–84, 01 2015. Disponível em: <http://dx.doi.org/10.1016/jjss.2015.01.027>.

[58] DATTA, S. K.; BONNET, C. Easing iot application development through datatweet framework. In: . [s.n.], 2016. p. 430–435. Disponível em: <http://dx.doi.org/10.1109/WF-IoT.2016.7845390>.

[59] KUM, S. W.; KANG, M.-g.; PARK, J.-I. Iot delegate: Smart home framework for heterogeneous iot service collaboration. *KSII Transactions on Internet and Information Systems*, v. 10, p. 3958–3971, 08 2016. Disponível em: <http://dx.doi.org/10.3837/tiis.2016.08.029>.

[60] MANIONE, R. User centered integration of internet of things devices. In: . [S.l.: s.n.], 2017. p. 102461K.

[61] SILVA, C. de M. et al. Design and evaluation of a services interface for the internet of things. *Wireless Personal Communications*, v. 91, n. 4, p. 1711–1748, 2016. Disponível em: <http://dx.doi.org/10.1007/s11277-015-3168-6>.

[62] FERRERA, E. et al. Xmpp-based infrastructure for iot network management and rapid services and applications development. *Annals of Telecommunications*, 07 2017. Disponível em: <http://dx.doi.org/10.1007/s12243-017-0586-3>.

[63] EGíDIO, D. J. B.; AQUINO, G. S. de. Viot – a step towards easing the interoperability of iot-based applications. In: ____. [s.n.], 2019. p. 482–496. ISBN 978-3-030-24307-4. Disponível em: <http://dx.doi.org/10.1007/978-3-030-24308-1$_3$9>.

[64] RIBEIRO, C. F. C. et al. Protocolos de redundância de gateway aplicados em redes iot. *36º Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2018.

[65] CHUNG, K.; PARK, R. C. P2p cloud network services for iot based disaster situations information. In: . Springer, 2016. v. 9, n. 3, p. 566–577. ISBN 1936-6442. Disponível em: <http://dx.doi.org/10.1007/s12083-015-0386-3>.

[66] PIRES, P. F. et al. Plataformas para a internet das coisas. In: ____. [S.l.: s.n.], 2015.

[67] PARAMESWARAN, M.; SUSARLA, A.; WHINSTON, A. B. P2p networking: an information sharing alternative. *Computer*, IEEE, v. 34, n. 7, p. 31–38, 2001. Disponível em: <http://dx.doi.org/10.1109/2.933501>.

[68] ROCHA, J. et al. Peer-to-peer: Computaçao colaborativa na internet. In: *Minicursos do XXII Simposio Brasileiro de Redes de Computadores (SBRC 2004)*. [S.l.: s.n.], 2004.

[69] AMFT, T.; GRAFFI, K. A tale of many networks: Splitting and merging of chord-like overlays in partitioned networks. *Technology of Social Networks Group, Heinrich Heine University, Düsseldorf, Germany, Tech. Rep. TR-2017-001*, 2017.

[70] PLUG, U.; FORUM, P. *Understanding Universal Plug and Play*. 2000. Disponível em: <http://www.upnp.org/download/UPNP$_u$*nderstandingUPNP.doc*>.

[71] FERREIRA, H. G. C.; CANEDO, E. D.; SOUSA, R. T. de. A ubiquitous communication architecture integrating transparent upnp and rest apis. *International Journal of Embedded Systems*, Inderscience, v. 6, n. 2, p. 188–197, 2014. Disponível em: <http://dx.doi.org/10.1504/IJES.2014.063816>.

[72] THOMA, M. et al. On iot-services: Survey, classification and enterprise integration. In: IEEE. *2012 IEEE International Conference on Green Computing and Communications*. 2012. p. 257–260. ISBN 978-1-4673-5146-1. Disponível em: <http://dx.doi.org/10.1109/GreenCom.2012.47>.

[73] BANDYOPADHYAY, D.; SEN, J. Internet of things: Applications and challenges in technology and standardization. *Wireless personal communications*, Springer, v. 58, n. 1, p. 49–69, 2011. Disponível em: <http://dx.doi.org/10.1007/s11277-011-0288-5>.

[74] GRØNBÆK, I. Architecture for the internet of things (iot): Api and interconnect. In: IEEE. *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*. 2008. p. 802–807. ISBN 978-0-7695-3330-8. Disponível em: <http://dx.doi.org/10.1109/SENSORCOMM.2008.20>.

[75] GRØNBÆK, I.; NORD, M.; JAKOBSSON, S. Abstract service api for connected objects. *R&I Report*, v. 18, p. 2007–06, 2007.

[76] XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, IEEE, v. 10, n. 4, p. 2233–2243, 2014. ISSN 1551-3203. Disponível em: <http://dx.doi.org/ 10.1109/TII.2014.2300753>.

[77] KIM, Y.; LEE, S.; CHONG, I. Orchestration in distributed web-of-objects for creation of user-centered iot service capability. *Wireless personal communications*, Springer, v. 78, n. 4, p. 1965–1980, 2014. Disponível em: <http://dx.doi.org/10.1109/ICUFN.2013.6614920>.

[78] JAKL, M. Rest representational state transfer. technical report. Vienna University of Technology, 2008.

[79] CURBERA, F. et al. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, IEEE, v. 6, n. 2, p. 86–93, 2002. Disponível em: <http://dx.doi.org/10.1109/4236.991449>.

[80] PLUG, U.; FORUM, P. *UPnP Device Architecture 1.0*. 2008. Disponível em: <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf>.

[81] FOWLER, M. *Patterns of enterprise application architecture*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.

[82] MACHADO, P. L. et al. Detection of electronic anklet wearers' groupings throughout telematics monitoring. *ISPRS International Journal of Geo-Information*, MDPI AG, v. 6, n. 1, p. 31, Jan 2017. ISSN 2220-9964. Disponível em: <http://dx.doi.org/10.3390/ijgi6010031>.

[83] ALBUQUERQUE, R. de O.; VILLALBA, L. J. G.; KIM, T.-H. Gtrust: group extension for trust models in distributed systems. *International Journal of Distributed Sensor Networks*, SAGE Publications Sage UK: London, England, v. 10, n. 2, p. 872842, 2014. Disponível em: <http://dx.doi.org/10.1155/2014/872842>.

[84] LEGENDRE, F.; AMORIM, M. D. de; FDIDA, S. Implicit merging of overlapping spontaneous networks [mobile ad hoc networks]. In: IEEE. *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*. 2004. v. 4, p. 3050–3054. ISBN 1090-3038. Disponível em: <http://dx.doi.org/10.1109/VETECF.2004.1400621>.

[85] BUIATI, F.; PUTTINI, R.; SOUSA, R. D. A secure autoconfiguration protocol for manet nodes. In: SPRINGER. *International Conference on Ad-Hoc Networks and Wireless*. 2004. p. 108–121. ISBN 0302-9743. Disponível em: <http://dx.doi.org/10.1007/978-3-540-28634-99>.

[86] FERREIRA, H. et al. Proposal of a secure, deployable and transparent middleware for internet of things. In: . [s.n.], 2014. Disponível em: <http://dx.doi.org/10.1109/CISTI.2014.6877069>.

[87] BRAINWY. *PyDev*. 2014–2018. Disponível em: <https://www.pydev.org/, Acessado em Março de 2019>.

[88] PRECORD, C. *wxPython 2.8 Application Development Cookbook*. [S.l.]: Packt Publishing Ltd, 2010.

[89] SMART, J.; CSOMOR, S. et al. *Cross-platform GUI programming with wxWidgets*. [S.l.]: Prentice Hall Professional, 2005.

[90] LORD, D. Flask is developed. *http://flask.pocoo.org/docs/1.0/license/authors. Acesso em: 18 de maio de 2019.*, 2010.

[91] FERREIRA, H. G. C. Arquitetura de middleware para internet das coisas. 2014. Disponível em: <http://repositorio.unb.br/handle/10482/17251>.

Apêndices

dht_client.py

```
import argparse
import socket
import sys
import threading
import hashlib
import os
import click

class client:
    def __init__(self, cname, cport, rname, rport):
        self.cn = cname
        self.cp = cport
        self.rn = rname
        self.rp = rport

    def listensocket(self):
        global sock
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        except socket.error as msg:
            print('Failed to create socket. Error code: ' + str(msg))
            sys.exit()
        try:
            sock.bind((self.cn, self.cp))
        except socket.error as msg:
            print('Bind failed. Error Code : ' + str(msg))
            sys.exit()
        sock.listen(10)

        while 1:
            conn, addr = sock.accept()
            req = conn.recv(1024)
            req = req.decode('utf-8')
            reqpro = req.split('|')
            if not req:
                break
            elif (reqpro[0] == "STORE") and (reqpro[1] == "RESP"):
```

```
38                  # Handles response with address of node to store
                    # STORE|RESP|KEY|NODENAME|NODEPORT
40                  nname = reqpro[3]
                    nport = int(reqpro[4])
42                  key = reqpro[2]
                    f1 = keystore[key]
44                  f1name = os.path.basename(f1)
                    f1read = open('dht', 'r')
46                  f1val = f1read.read()
                    data2 = "STORE|OBJ|"+key+"|"+f1name+"|"+f1val
48                  f1read.close()
                    sendrequest(nname, nport, data2)
50
                elif (reqpro[0] == "ITER"):
52                  # Display  and store the object returned after iterative
    query
                    # ITER|YES|NODENAME|NODEPORT|KEY|OBJECTNAME|OBJECTVALUE
54                  key = reqpro[4]
                    print(key)
56                  objectname = reqpro[5]
                    objectvalue = reqpro[6]
58                  objfile = open('dht', 'a')
                    objfile.write(objectname + ':' + key+'\n')
60                  objfile.close
                    print("Object retreived by iteratively querying the CHORD
    peers\n")
62


64
def menuopt(node):
66      while 1:
            menu_opt = input(
68              "1. Salvar um objeto \n2. Recuperar um objeto de um n  \n3. Sair-
    e\n ")

70          if menu_opt == "1":
                filepath = input(
72                  "Insira o caminho do arquivo:\nExemplo: /home/user/filename.
    txt\n")
                filename = os.path.basename(filepath)
74              key = hashlib.sha1(filename.encode('utf-8')).hexdigest()
                keystore[key] = filepath
76              store_lookup = "STORE|OBJ|"+key+"|"+dclient.cn+"|"+str(dclient.cp
    )
                sendrequest(dclient.rn, dclient.rp, store_lookup)
78              print("Store  request sent")
            elif menu_opt == "2":
80              keyval = input(
                    "Enter the key value of the object to be retreived.")
```

```
            keyval = hashlib.sha1(keyval.encode('utf-8')).hexdigest()
            iter_lookup = "RETREIVE|ITER|"+keyval + \
                "|"+dclient.cn+"|"+str(dclient.cp)
            sendrequest(dclient.rn, dclient.rp, iter_lookup)

        elif menu_opt == "3":
            print("Saindo...")
            sys.exit()
        else:
            print("Comando inv lido")
            sys.exit()


def sendrequest(remotehost, remoteport, senddata):
    sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    remote_ip = socket.gethostbyname(remotehost)
    sock2.connect((remote_ip, remoteport))
    sock2.sendall(senddata.encode('utf-8'))
    sock2.close()


@click.group(invoke_without_command=True)
@click.option('-p', '--client_port', type=int,
              help='Specify the port for the peer')
@click.option('-h', '--client_hostname',
              help='Specify the hostname of the peer')
@click.option('-r', '--root_port', type=int,
              help='Specify the port of the root')
@click.option('-R', '--root_hostname',
              help='Specify the hostname of the root')
def start(client_port, client_hostname, root_port, root_hostname):
    global keystore
    keystore = {}

    dclient = client(client_hostname, client_port, root_hostname, rootport)

    sockcl = threading.Thread(target=dclient.listensocket)
    menu = threading.Thread(target=menuopt, args=(dclient,))
    sockcl.start()
    menu.start()
```

code/dht_client.py

dht_peer.py

```
from raise_p2p.p2p.node import Node

import threading
import struct
import socket
```

```
import time
import json

SHOW_MESSAGE = "http://localhost:5000/message"


class Peer(Node):
  """Peer node of a P2P network.

  IPv4 (AF_INET) protocol with TCP (SOCK_STREAM).

  Args:
    host_ip (str): Host IP from host name.
  """
  PORT = 12666

  def __init__(self, host_ip, socketio):
    self.HOST = host_ip
    self.stop = False
    self.socketio = socketio
    self.socketio.emit('message', {'data': 'Peer iniciado.'},
              namespace='/message')
    self.node = Node(host_ip)

    self.commands = {'BACKUP': self.backup,
            'RESTORE': self.restore,
            'BACKUP_RCV': self.backup_rcv,
            'RESTORE_RCV': self.restore_rcv}

    self.init_socket()

  def init_socket(self):
    """Start TCP server socket on port 12666.

    Queue up 10 connect requests before refusing outside connections.
    """
    #
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # SO_REUSEADDR : the port 12666 will be immediately reusable after
    # the socket is closed
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.bind((self.HOST, self.PORT))
    self.socket.listen(10)
    self.socketio.emit('message', {'data': 'Socket iniciado.'},
              namespace='/message')

  def run(self):
    while not self.stop:
      client_conn, client_addr = self.socket.accept()
```

```
56          self.socketio.emit('ip_client', str(client_addr[0]),
                    namespace='/client')
58
            # Receive data in small chunks and retransmit it
60          rcv_data = threading.Thread(target=self.handle_client,
                        args=(client_conn, client_addr))
62          rcv_data.start()

64      self.close()

66  def handle_client(self, sckt, address):
        peer_name = sckt.getpeername()
68      msg = f'Connected {peer_name}'
        self.socketio.emit('message', {'data': msg}, namespace='/message')
70
        while True:
72          # receive command, then data from backup or restore
            data_rcv = self.read(sckt)
74          if 'command' in data_rcv:
                data = data_rcv['command']
76              msg = f'Mensagem recebida : {data}'
                self.socketio.emit('message', {'data': msg},
78                      namespace='/message')
                # send data to backup_rcv or restore_rcv
80              self.commands.get(data.upper(),
                    lambda x: print(f'Command {data} not found.'))(sckt)
82
        sckt.close()
84
        print(f'Disconnecting {peer_name}')
86
    def send_to_peer(self, host, command):
88      sckt = self.peer_connection(host)

90      # send data from backup or restore
        self.commands.get(command.upper(),
92          lambda x: print(f'Command {command} not found.'))(sckt)

94      sckt.close()

96  def backup_rcv(self, sckt):
        print(f'Peer try to read from backup_rcv...')
98      data_rcv = self.read(sckt)

100     if not data_rcv:
            msg = 'DADOS N O FORAM RECEBIDOS'
102     else:
            msg = 'DADOS RECEBIDOS DO BACKUP'
```

```
104         self.socketio.emit('backup_client', {'data': data_rcv},
                    namespace='/client')
106
      print(msg)
108     self.socketio.emit('message', {'data': msg}, namespace='/message')
110
    def backup(self, sckt):
112     msg = 'CARREGANDO DADOS DO BACKUP...'
      print(msg)
114     self.socketio.emit('message', {'data': msg}, namespace='/message')
116     if self.send(sckt, {'command': 'BACKUP_RCV'}):
        print(f'Peer sent BACKUP_RCV...')
118
      if self.node.data and self.send(sckt, self.node.data):
120       msg = 'DADOS ENVIADOS COM SUCESSO'
      else:
122       msg = 'DADOS N O FORAM ENVIADOS..'
124     print(msg)
      self.socketio.emit('message', {'data': msg}, namespace='/message')
126
    def restore_rcv(self, sckt):
128     print(f'Peer try to read from restore_rcv...')
      data_rcv = self.read(sckt)
130
      if not data_rcv:
132       msg = 'DADOS N O FORAM RECEBIDOS'
      else:
134       msg = 'DADOS RECEBIDOS DO RESTORE'
        self.socketio.emit('restore_client', {'data': data_rcv},
136                 namespace='/client')
138     print(msg)
      self.socketio.emit('message', {'data': msg}, namespace='/message')
140
    def restore(self, sckt):
142     msg = 'CARREGANDO DADOS...'
      self.socketio.emit('message', {'data': msg}, namespace='/message')
144
      if self.send(sckt, {'command': 'RESTORE_RCV'}):
146       print(f'Peer sent RESTORE_RCV...')
148     ip, port = sckt.getpeername()
      data = self.node.data_by_ip(ip=ip)
150
      if self.send(sckt, data) and data:
152       msg = 'DADOS DO RESTORE ENVIADOS!'
```

```
154         print(msg)
            self.socketio.emit('message', {'data': msg}, namespace='/message')
156
            if self.node.delete_by_ip(ip=ip):
158           msg = f'DADOS DO IP {ip} REMOVIDOS'
            else:
160           msg = f'N O  H  DADOS PARA REMOVER DO {ip}'
          else:
162         print("Peer couldn't send restore...")
            msg = 'DADOS N O FORAM ENVIADOS NEM REMOVIDOS..'
164
          print(msg)
166       self.socketio.emit('message', {'data': msg}, namespace='/message')

168     def peer_connection(self, host, sock=None):
          if not sock:
170         client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            not_connect = True
172
            while not_connect:
174           try:
                client_socket.connect((host, self.PORT))
176           except:
                print('Trying to connect again...')
178             time.sleep(1)
              else:
180             not_connect = False
          else:
182         client_socket = sock

184       return client_socket

186     def send(self, sckt, data):
          try:
188         serialized = json.dumps(data)
          except (TypeError, ValueError) as e:
190         raise Exception('You can only send JSON-serializable data')
          # send the length of the serialized data first
192       sckt.send(b'%d\n' % len(serialized))
          # send the serialized data
194       sckt.sendall(serialized.encode())

196       return True

198     def read(self, sckt):
          # read the length of the data, letter by letter until we reach EOL
200       length_str = ''
          char = sckt.recv(1).decode()
```

```
202
    while char != '\n':
204     length_str += char
        char = sckt.recv(1).decode()
206   total = int(length_str)

208   # use a memoryview to receive the data chunk by chunk efficiently
    view = memoryview(bytearray(total))
210   next_offset = 0

212   while total - next_offset > 0:
        recv_size = sckt.recv_into(view[next_offset:], total - next_offset)
214     next_offset += recv_size
    try:
216     deserialized = json.loads(view.tobytes())
    except (TypeError, ValueError) as e:
218     raise Exception('Data received was not in JSON format')

220   return deserialized

222 def close(self):
    """Close TCP server socket on port 12666.
224 """
    self.socket.close()
```

code/peer.py

node.py

```
from raise_p2p import query
2


4 class Node:
  def __init__(self, host_ip):
6   self.data = {'clients': [], 'services': [], 'datas': [], 'tokens': []}

8   self.update_data()

10 def update_data(self):
    self.data['clients'] = query.get_client()[0]
12  self.data['services'] = query.get_service()[0]
    self.data['datas'] = query.get_data()[0]
14  self.data['tokens'] = query.get_token()[0]

16 def data_by_ip(self, ip):
    tmp_data = {'clients': [], 'services': [], 'datas': [], 'tokens': []}
18  tmp_data['clients'] = query.get_client(ip=ip)[0]
    tmp_data['services'] = query.get_service(ip=ip)[0]
20  tmp_data['datas'] = query.get_data(ip=ip)[0]
    tmp_data['tokens'] = query.get_token(ip=ip)[0]
```

```
22
    return tmp_data
24
  def delete_by_ip(self, ip):
26    return query.delete_data(ip=ip)
```

code/node.py

app.py

```
from flask import Flask, Response, render_template, jsonify, request, abort,\
2                   Blueprint, g
from .validation import JsonValidation, ArgsValidation
4 from .report import LogReport
from pymongo import errors
6 import logging
import json
8 import sys
import configparser
10 from bson import ObjectId
import datetime
12 import time

14 # Fun  o que substitui o aggregate de service
def service_aggregate():
16     responses = list(g.db.service_collection.find())

18     for service in responses:
        datas = list(g.db.data_collection.aggregate([
20                 {
                       "$match":
22                     {
                           'mac':service['mac'],
24                         'chipset':service['chipset']
                       }
26                 }
        ]))
28
        total = 0
30      for data in datas:
            total += data['size']
32
        last_update_date = datas[-1]["_id"].generation_time.strftime("%a - %d
    %b %Y %H:%M:%S")
34      service['last-update-data'] = last_update_date
        del service['_id']
36
        service['size'] = total
38
    return responses
```

```
40
42 api = Blueprint('api', __name__)
44
46 @api.route('/', methods=['GET'])
   def index():
48     return 'DIMS has been started'
50 @api.route('/client', methods=['POST'])
   def client_register():
52     client = request.get_json() if request.is_json else request.form.to_dict
       ()
       if not JsonValidation().client_validation(client):
54         abort(400)
       try:
56         g.db.client_collection.update_one(client, {'$set': client},
                                             upsert=True)
58     except errors.WriteError:
           return jsonify({'code': 500, 'message': 'Internal Server Error'}),
       500
60     return jsonify({'code': 200, 'message': 'Success'}), 200
62
   @api.route('/service', methods=['POST'])
64 def service_register():
       service = request.get_json() if request.is_json else request.form.to_dict
       ()
66     if not JsonValidation().service_validation(service):
           abort(400)
68
       if not g.db.client_collection.find_one({'chipset': service['chipset'],
70                                             'mac': service['mac']}):
           abort(422, {'message': 'Client not found',
72                     'details': 'The client passed in service body could '
                      'not be found. Check the chipset and mac values.'})
74     try:
           g.db.service_collection.update_one(service, {'$set': service},
76                                             upsert=True)
       except errors.WriteError:
78         return jsonify({'code': 500, 'message': 'Internal Server Error'}),
       500
       return jsonify({'code': 200, 'message': 'Success'}), 200
80
82 @api.route('/data', methods=['POST'])
   def data_register():
84     data = request.get_json() if request.is_json else request.form.to_dict()
```

```
    if not JsonValidation().data_validation(data):
        abort(400)

    if 'counter' in data:
        if data['counter'] == -1:
            LogReport().end_report()
        logging.getLogger('dims').debug(f'"{data}"',
                                        extra={'size': sys.getsizeof(data)})

    client = g.db.client_collection.find_one({'chipset': data['chipset'],
                                              'mac': data['mac']})
    service = g.db.service_collection.find_one({'chipset': data['chipset'],
                                                'mac': data['mac'], 'number':
                                                data['serviceNumber']})

    if not (client and service):
        abort(422, {'message': 'Client or Service not found',
                    'details': 'The client or service passed in data body '
                    'could not be found. Check the chipset, mac and '
                    'serviceNumber values.'})

    data_size = sys.getsizeof(data)

    data['size'] = int(data_size)
    data['type'] = 'uncompressed'
    data['time_insert'] = float(datetime.datetime.now().timestamp())
    last_inserted_data = list(g.db.data_collection.find().sort([('_id',-1)]).
    limit(1))
    if len(last_inserted_data) != 0:
        data['time_diff'] = float(data['time_insert'])-float(
    last_inserted_data[0]['time_insert'])
    else:
        data['time_diff'] = 0
    try:
        g.db.data_collection.insert_one(data)
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200


@api.route('/list/client', methods=['GET'])
def client_list():
    amount = request.args.get('latest', default=0, type=int)

    response = list(g.db.client_collection.find(ArgsValidation().client_args(
                                                request.args)).sort(
                                                "$natural", -1).limit(amount)
    )
```

```
130
    for document in response:
132         del document['_id']

134     return jsonify(response), 200

136 # ---------- Se  o de listagem de clientes contendo mais informa  es
    como last-update-time and number of services ----------#
@api.route('/list/client_full', methods=['GET'])
138 def client_list_full():
    amount = request.args.get('latest', default=0, type=int)
140     pipeline = [

142         { '$lookup':
                    {
144                     'from' : 'datas',
                        'localField' : 'chipset',
146                     'foreignField' : 'chipset',
                        'as' : 'data-aggregation'
148                 }
        }
150     ]
    table_agregation = list(g.db.client_collection.aggregate(pipeline))
152
    for document in table_agregation:
154         last_update_date = ""
        if(document["data-aggregation"] != []):
156             last_update_date = document["data-aggregation"][-1]["_id"].
    generation_time.strftime("%d/%m/%y %H:%M:%S")

158         count_services = g.db.service_collection.count({"mac": document["mac"
    ], "chipset": document["chipset"]})
        document['last-update-data'] = last_update_date
160         document['number-of-services'] = count_services
        del document['_id']
162         del document['data-aggregation']


164
    return jsonify(table_agregation), 200
166 # ---------- Finaliza  o de listagem contendo mais informa  es
    ---------- #

168 # ---------- Se  o de cria  o de regras provis rio ----------#
@api.route('/list/service')
170 def service_list():
    amount = request.args.get('latest', default=0, type=int)
172
    response = service_aggregate()
174
```

```
    return jsonify(response), 200


@api.route('/list/data')
def data_list():
    amount = request.args.get('latest', default=0, type=int)

    response = list(g.db.data_collection.find(ArgsValidation().data_args(
                                                request.args)).sort(
                                                "$natural", -1).limit(amount))


    for document in response:
        del document['_id']

    return jsonify(response), 200


@api.route('/report', methods=['GET'])
def report_data():
    LogReport().init_report_config()
    return jsonify({'code': 200, 'message': 'Success'}), 200


# ---------- Se  o de cria  o de regras provis rio ----------#

@api.route("/rule", methods=['POST'])
def insert_rule():
    rule = request.get_json() if request.is_json else request.form.to_dict()

    try:
        g.db.rule_collection.insert_one(rule)
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200

@api.route('/list/rule', methods=['GET'])
def list_rule():

    try:
        response = list(g.db.rule_collection.find(ArgsValidation.rule_args(
                                                    request.args )))
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500

    for document in response:
        del document['_id']
```

```
222
    return jsonify(response), 200
224
# ---------- Finaliza  o de cria  o de regras ---------- #
226


228
# ---------- List Interfaces (API para tela inicial de todos os
    projetos / gateway local) ---------- #
230
@api.route('/list/interface', methods=['GET'])
232 def list_interface():
    try:
234     # ---------- Contagem de quantidade de clientes ----------
    #

236     channel_list = ['HTTP','ZIGBEE','TCP','UDP']
        list_response = []
238     # Aggregation que calcula a quantidade de clientes que usam os
    protocolos especificados na lista channel_list
        for channel in channel_list:
240         json_reponse = {}
            json_reponse["name"] = channel
242         json_reponse["clients"] = 0
            json_reponse["services"] = 0
244         json_reponse["data_count"] = 0
            pipeline_clients_services = [
246             {

248                 '$match':
                    {
250                     'channel':channel
                    }
252             }
                ,
254             {
                    '$count':"count"
256             }
            ]
258
            pipeline_data = [
260             {
                    '$match':
262                 {
                        'channel':channel
264                 }
                },
266             {
                    '$group':
```

```
268                     {
                            '_id':'$channel',
270                         'channel_bytes_sum':{'$sum':'$size'}
                        }
272                 }
            ]
274
            client_list = list(g.db.client_collection.aggregate(
    pipeline_clients_services))
276         if len(client_list) != 0:
                temp_channel = channel.lower()+"_clients"
278             json_reponse["clients"] = client_list[0]['count']

280     # Aggregation que calcula a quantidade de servi os por mecanismo de
    comuni  o utilizado na lista lista
            service_list = list(g.db.service_collection.aggregate(
    pipeline_clients_services))
282         if len(service_list) != 0:
                temp_channel = channel.lower()+"_service"
284             json_reponse["services"] = service_list[0]['count']


286
        # Aggregation que calcula a quantidade de dados que usam o mecanismo
    de comunica  o utilizado na lista
288         data_list = list(g.db.data_collection.aggregate(pipeline_data))
            if len(data_list) != 0:
290             temp_channel = channel.lower()+"_data"
                json_reponse["data_count"] = data_list[0]['channel_bytes_sum'
    ]
292
            list_response.append(json_reponse)
294
     except errors.WriteError:
296         return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500

298     return jsonify(list_response), 200


300
# ------------------------------ Fim do list interface
    ------------------------------ #
302


304 # ------------------------------ List Gateway (listar informa  es acerca de
    gateways ) ------------------------------ #

306 @api.route('/list/gateway',methods=['GET'])
    def list_gateway():
308     # Inserir a coleta dos dados associados ao gateway por meio da API que a
```

```
    Daiane est  fazendo

    # Realiza a contagem de clientes/servi os/rules
    client_count = g.db.client_collection.count()
    service_count = g.db.service_collection.count()
    rule_count = g.db.rule_collection.count()

    time_now = ObjectId.from_datetime(datetime.datetime.now()+datetime.
timedelta(minutes=180))
    time_now_minus_5 = ObjectId.from_datetime(datetime.datetime.now()+
datetime.timedelta(minutes=180)-datetime.timedelta(minutes=5))

    compressed_data_count = list(g.db.data_collection.aggregate([
        {
            "$match":
            {
                'type':'compressed',
                 '_id':{'$gt':time_now_minus_5}
            }
        }
        ,
        {
           '$group':
           {
               '_id':'$type',
               'count':{'$sum':1},
               'sum_size_compressed':{'$sum':'$size_compressed'},
               'sum_size_document':{'$sum':'$size'}
           }
        }

        ]))

    # Aggregate que soma o tamanho dos tipos compressed e uncompressed e
retorna em chaves de nome diferente
    rate_sum = list(g.db.data_collection.aggregate([
        {
            "$match":
            {
                '_id':{'$gt':time_now_minus_5}
            }
        }
        ,
        {
            '$project':
            {
                '_id':'compressed',
                'compressed_sum':{
                    '$cond':{'if':{'$eq':['$type','compressed']},'then':{'
```

```
    $sum':'$size_compressed'},'else':{'$sum':0}}
354                 },
                    'uncompressed_sum':{
356                     '$cond':{'if':{'$eq':['$type','uncompressed']},'then':{'
    $sum':'$size'},'else':{'$sum':0}}
                    }
358              }
         }
         ,
         {
362             '$group':
                {
364                 '_id':'compressed',
                    'compressed_sum':{'$sum':'$compressed_sum'},
366                 'uncompressed_sum':{'$sum':'$uncompressed_sum'}
                }
368      }
         ]))
370
372      # Aggregate que pega o documento mais recente da data_collection
         collections = [g.db.data_collection, g.db.service_collection, g.db.
    client_collection]
374      time_dict = {}
         for collection in collections:
376          latest_time = list(collection.aggregate([
                {
378                 '$sort':{'_id': -1}
                },
380             {
                    '$limit': 1
382             }
                ,
384             {
                    '$project':{
386                     '_id':'$_id'
                    }
388             }
         ]))
390      #print(latest_time[0]['_id'].generation_time)
         #print(latest_time[0]['_id'])
392      if len(latest_time) != 0:
             iso_timestamp = latest_time[0]['_id'].generation_time
394          time_value = time.mktime(iso_timestamp.timetuple())
             if(collection == g.db.data_collection):
396              time_dict['data'] = time_value
                 time_dict['data_utc'] = iso_timestamp
398          elif(collection == g.db.service_collection):
                 time_dict['service'] = time_value
```

```
400                 time_dict['service_utc'] = iso_timestamp
            elif(collection == g.db.client_collection):
402                 time_dict['client'] = time_value
                    time_dict['client_utc'] = iso_timestamp
404         else:
                time_dict['data'] = 0
406             time_dict['service'] = 0
                time_dict['client'] = 0
408
    if time_dict['data'] == 0 and time_dict['service'] == 0 and time_dict['
    client'] == 0:
410         final_time = time_dict['data_utc']

412     type_list = ['client','service','data']
    biggest_time = 0
414     biggest_time_utc = ""
    for typeof in type_list:
416         if (time_dict[typeof] > biggest_time):
                biggest_time = time_dict[typeof]
418             biggest_time_utc = time_dict[typeof+"_utc"]

420     if len(rate_sum) != 0:
            compressed_sum = rate_sum[0]['compressed_sum']
422         uncompressed_sum = rate_sum[0]['uncompressed_sum']

424         compressed_rate = compressed_sum/300
            uncompressed_rate = uncompressed_sum/300
426     else:
            compressed_rate = 0
428         uncompressed_rate = 0



432     if len(compressed_data_count) != 0:
            # Dados sobre o batch de dados enviados, compactados ou n o
434         count = int(compressed_data_count[0]['count'])
            sum_size_compressed = int(compressed_data_count[0]['
    sum_size_compressed'])
436         sum_size_document = int(compressed_data_count[0]['sum_size_document'
    ])

438         # Calculando o percentual de utiliza   o do batch size
            compression_rate = (sum_size_compressed/sum_size_document)
440         percent = (1-compression_rate)*100
    else:
442         percent = 0

444     response = {
            'percent':round(percent,2),
```

```
446             'client_count':client_count,
                'service_count':service_count,
448             'rule_count':rule_count,
                'compacted_rate':round(compressed_rate,2),
450             'not_compacted_rate':round(uncompressed_rate,2),
                'last_updated':biggest_time_utc,
452             'gateway_name':0,
                'gateway_master':0
454     }
    response_list = []
456     response_list.append(response)

458     return jsonify(response_list),200


460


462 # ------------------------------ Fim de list gateway
    ---------------------------------- #

464 # ---------------------------- List service-cloud (listar informa  es acerca
    de service-cloud ) ------------------------- #

466 @api.route('/list/service-cloud', methods=['GET'])
def service_cloud():
468

    # Pega todos os clientes existentes na collection
470     amount = request.args.get('latest', default=0, type=int)
    clients = list(g.db.client_collection.find(ArgsValidation().client_args(
472                                                 request.args)).sort(
                                                 "$natural", -1).limit(amount)
    )
474

    service_list = service_aggregate()
476

    response_list = []
478     for client in clients:

        for service in service_list:

482             response_json = {
                    'device_name':'None',
484                 'service_name':'None',
                    'value':0,
486                 'associate_rules':0,
                    'data_amount':0,
488                 'critical_messages':0,
                    'storage_time':0,
490                 'last_updated':'None'
                }
```

```
492
            if(service['mac'] == client['mac'] and service['chipset'] ==
    client['chipset']):
494
                ## Aggregate para pegar todas as mensagens cr ticas
    associadas a um determinado servi o
496             data_list = list(g.db.data_collection.aggregate([
                {
498                 "$match":
                    {
500                     'sensitive':'1',
                        'chipset':service['chipset'],
502                     'mac':service['mac']
                    }
504             }
                ]))
506
                ## Aggregate para pegar todas as mensagens associadas as um
    determinado servi o
508             data_listed = list(g.db.data_collection.aggregate([
                {
510                 "$match":
                    {
512                     'chipset':service['chipset'],
                        'mac':service['mac']
514                 }
                }
516             ]))

518             ## Aggregate para pegar todas as regras associadas a um
    determinado servi o
                rule_response = list(g.db.rule_collection.aggregate([
520             {
                    "$match":
522                 {
                        'chipset':service['chipset'],
524                     'mac':service['mac']
                    }
526             }
                ,
528             {
                    "$project":
530                 {
                        "_id":"rule",
532                     "count":{"$sum":1}
                    }
534             }
                ]))
536
```

```
                # Coloca os valores adquiridos no dicion rio response_json
538             response_json['device_name'] = client['name']
                response_json['service_name'] = service['name']
540             if len(rule_response) != 0:
                    response_json['associate_rules'] = rule_response[0]['
    count']
542             response_json['data_amount'] = service['size']
                response_json['critical_messages'] = len(data_list)
544             response_json['last_updated'] = service['last-update-data']
                try:
546                 response_json['value'] = str(data_listed[-1]['value'
    ][-1])
                except IndexError:
548                 response_json['value'] = 'None'

550             # Coloca o dicion rio na lista final de resposta
                response_list.append(response_json)
552
    return jsonify(response_list),200
554
# ---------- Fim do list service-cloud ---------- #
556

558 # ---------- Listagem service-cloud-interface ---------- #
@api.route('/list/service-cloud-interface',methods=['GET'])
560 def service_cloud_interface():

562     # Pega todos os clientes existentes na collection
    amount = request.args.get('latest', default=0, type=int)
564     clients = list(g.db.client_collection.find(ArgsValidation().client_args(
                                                request.args)).sort(
566                                                "$natural", -1).limit(amount)
    )


568
    response_list = []
570     for client in clients:

572         ## JSON que mostra o formato da resposta do m todo da API
            response_json = {
574             'device_name':'None',
                'device_interface':'None',
576             'device_identifier':0,
                'service_count':0,
578             'last_updated':'None',
                'register_expiration':'None'
580         }

582         ## Aggregate que pega todos os dados associados ao cliente da
```

```
    itera   o atual
        data_response = list(g.db.data_collection.aggregate([
584             {
                    "$match":
586                 {
                        'chipset':client['chipset'],
588                     'mac':client['mac']
                    }
590             }
        ]))
592     ## Pega o  ltimo  elemento da lista retornada e coloca em
    last_data_update o hor rio em que
        ## o documento foi inserido
594     if(len(data_response) != 0):
            last_data_update = data_response[-1]['_id'].generation_time.
    strftime("%a - %d %b %Y %H:%M:%S")
596     service_count = g.db.service_collection.count()
        response_json['device_name'] = client['name']
598     response_json['device_interface'] = client['channel']
        response_json['device_identifier'] = client['mac']
600     response_json['service_count'] = service_count
        response_json['last_updated'] = last_data_update
602
        response_list.append(response_json)
604
    return jsonify(response_list),200
606
# ------------------ FIM Listagem service-cloud-interface ---------------- #
608
# ------------------ Listagem service-cloud-average
   ------------------------ #
610 @api.route('/list/service-cloud-average',methods=['GET'])
def service_cloud_average():
612
    # Pega todos os clientes existentes na collection
614 amount = request.args.get('latest', default=0, type=int)
    clients = list(g.db.client_collection.find(ArgsValidation().client_args(
616                                             request.args)).sort(
                                                "$natural", -1).limit(amount)
    )
618
    service_list = service_aggregate()
620
    result_list = []
622 for client in clients:

624     for service in service_list:

626         # JSON que contem o formato da resposta do m todo
```

```
            response_json = {
628             'device_name':'None',
                'service_name':'None',
630             'input_average':0,
                'last_updated':'None'
632         }

634         # Aggregate que pega todos os dados associados a um servi o e
    calcula a m dia de time_diff
            data_time_average = list(g.db.data_collection.aggregate([
636             {
                    "$match":
638                 {
                        'serviceNumber':service['number']
640                 }
                }
642             ,
                {
644                 "$group":
                    {
646                     '_id':'average',
                        'average_input_time':{'$avg':'$time_diff'}
648                 }
                }
650         ]))


652
            if(service['mac'] == client['mac'] and service['chipset'] ==
    client['chipset']):
654             response_json['device_name'] = client['name']
                response_json['service_name'] = service['name']
656             response_json['last_updated'] = service['last-update-data']
                if len(data_time_average) != 0:
658                 response_json['input_average'] = data_time_average[0]['
    average_input_time']

660             result_list.append(response_json)

662     return jsonify(result_list),200

664 # ---------------------------------------- M todo de inser   o de dados
    enviados ---------------------- #
@api.route('/telemetry',methods=['POST'])
666 def send_data_telemetry():
    telemetry_dict = request.get_json() if request.is_json else request.form.
    to_dict()
668
    try:
670     g.db.telemetry_collection.insert_one(telemetry_dict)
```

```
    except errors.WriteError:
672         return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200
674
# ---------------------------------- FIM do m todo de inser   o de dados
    enviados ------------------------ #
676


678 # ---------------------------------------- M todo de inser   o de dados
    enviados ---------------------- #
@api.route('/telemetry-gateway',methods=['POST'])
680 def send_data_telemetry_gateway():
    telemetry_dict = request.get_json() if request.is_json else request.form.
    to_dict()
682     gateway = list(g.db.telemetry_collection.find_one({'remote_gateway_name':
     telemetry_dict['remote_gateway_name']}))  if g.db.telemetry_collection.
    find_one({'remote_gateway_name': telemetry_dict['remote_gateway_name']})
    != None else []
    if len(gateway) != 0:
684         try:
                g.db.telemetry_collection.insert_one(telemetry_dict)
686         except errors.WriteError:
                return jsonify({'code': 500, 'message': 'Internal Server Error'})
    , 500
688         return jsonify({'code': 200, 'message': 'Success'}), 200
    else:
690         return jsonify({'code':400,'message':'remote-gateway already exists'
    }),400

692 # ---------------------------------- FIM do m todo de inser   o de dados
    enviados ------------------------ #

694 @api.route('/list/gateway-cloud',methods=['GET'])
def list_gateway_cloud():
696
    # Gera o ObjectID de agora e 5 minutos atr s
698     time_now = ObjectId.from_datetime(datetime.datetime.now()+datetime.
    timedelta(minutes=180))
    time_now_minus_5 = ObjectId.from_datetime(datetime.datetime.now()+
    datetime.timedelta(minutes=180)-datetime.timedelta(minutes=5))
700
    # Pega todos os remote gateways cadastrados em telemetry_collection
702     remote_gateways = list(g.db.telemetry_collection.find({'
    remote_gateway_name':{'$exists':'true'}}))

704     print("REMOTE GATEWAYS: {}".format(remote_gateways))

706     response_list = []
```

```
    for remote_gateway in remote_gateways:

        response = {
            'remote_gateway':'None',
            'total_sent_compacted':0,
            'economy':0,
            'sensitive_transmission':0,
            'compacted_transmission':0
        }

        uncompressed_data = list(g.db.telemetry_collection.aggregate([
        {
            "$match":
            {
                'type':'uncompressed',
                 '_id':{'$gt':time_now_minus_5},
                 'remote_gateway':remote_gateway['remote_gateway_name']
            }
        }
        ,
        {
           '$group':
           {
               '_id':'$type',
               'count':{'$sum':1},
               'sum_size':{'$sum':'$sent_size'},
           }
        }

        ]))

        compressed_data = list(g.db.telemetry_collection.aggregate([
        {
            "$match":
            {
                'type':'compressed',
                 '_id':{'$gt':time_now_minus_5},
                 'remote_gateway':remote_gateway['remote_gateway_name']
            }
        }
        ,
        {
           '$group':
           {
               '_id':'$type',
               'count':{'$sum':1},
               'sum_size':{'$sum':'$sent_size'},
           }
        }
```

```
        ]))

        sensitive_transmission = list(g.db.telemetry_collection.aggregate([
        {
            "$match":
            {
                'senstive':1,
                '_id':{'$gt':time_now_minus_5},
                'remote_gateway':remote_gateway['remote_gateway_name']
            }
        }
        ,
        {
           '$group':
           {
               '_id':'$type',
               'count':{'$sum':1},
               'sum_size':{'$sum':'$sent_size'},
           }
        }
        ]))

        if len(compressed_data) != 0:
            response['compacted_transmission'] = compressed_data[0]['sum_size
    ']/300
            response['total_sent_compacted'] = compressed_data[0]['sum_size']

        if len(sensitive_transmission) != 0:
            response['sensitive_transmission'] = sensitive_transmission[0]['
    sum_size']/300

        response['remote_gateway'] = remote_gateway['remote_gateway_name']

        response_list.append(response)

    return jsonify(response_list),200
```

code/app.py

app.py

```
from flask import Flask, Response, render_template, jsonify, request, abort,\
                  Blueprint, g
from .validation import JsonValidation, ArgsValidation
from .report import LogReport
from pymongo import errors
import logging
import json
import sys
import configparser
```

```
from bson import ObjectId
import datetime
import time

# Fun  o que substitui o aggregate de service
def service_aggregate():
    responses = list(g.db.service_collection.find())

    for service in responses:
        datas = list(g.db.data_collection.aggregate([
                {
                    "$match":
                    {
                        'mac':service['mac'],
                        'chipset':service['chipset']
                    }
                }
        ]))

        total = 0
        for data in datas:
            total += data['size']

        last_update_date = datas[-1]["_id"].generation_time.strftime("%a - %d %b %Y %H:%M:%S")
        service['last-update-data'] = last_update_date
        del service['_id']

        service['size'] = total

    return responses


api = Blueprint('api', __name__)


@api.route('/', methods=['GET'])
def index():
    return 'DIMS has been started'


@api.route('/client', methods=['POST'])
def client_register():
    client = request.get_json() if request.is_json else request.form.to_dict()
    if not JsonValidation().client_validation(client):
        abort(400)
    try:
        g.db.client_collection.update_one(client, {'$set': client},
```

```
                                            upsert=True)
58      except errors.WriteError:
            return jsonify({'code': 500, 'message': 'Internal Server Error'}),
        500
60      return jsonify({'code': 200, 'message': 'Success'}), 200


62
@api.route('/service', methods=['POST'])
64  def service_register():
        service = request.get_json() if request.is_json else request.form.to_dict
        ()
66      if not JsonValidation().service_validation(service):
            abort(400)
68
        if not g.db.client_collection.find_one({'chipset': service['chipset'],
70                                                'mac': service['mac']}):
            abort(422, {'message': 'Client not found',
72                      'details': 'The client passed in service body could '
                        'not be found. Check the chipset and mac values.'})
74      try:
            g.db.service_collection.update_one(service, {'$set': service},
76                                               upsert=True)
        except errors.WriteError:
78          return jsonify({'code': 500, 'message': 'Internal Server Error'}),
        500
        return jsonify({'code': 200, 'message': 'Success'}), 200
80


82  @api.route('/data', methods=['POST'])
    def data_register():
84      data = request.get_json() if request.is_json else request.form.to_dict()
        if not JsonValidation().data_validation(data):
86          abort(400)

88      if 'counter' in data:
            if data['counter'] == -1:
90              LogReport().end_report()
            logging.getLogger('dims').debug(f'"{data}"',
92                                            extra={'size': sys.getsizeof(data)})

94      client = g.db.client_collection.find_one({'chipset': data['chipset'],
                                                  'mac': data['mac']})
96      service = g.db.service_collection.find_one({'chipset': data['chipset'],
                                                    'mac': data['mac'], 'number':
98                                                  data['serviceNumber']})

100     if not (client and service):
            abort(422, {'message': 'Client or Service not found',
102                     'details': 'The client or service passed in data body '
```

```
                    'could not be found. Check the chipset, mac and '
                    'serviceNumber values.'})

    data_size = sys.getsizeof(data)

    data['size'] = int(data_size)
    data['type'] = 'uncompressed'
    data['time_insert'] = float(datetime.datetime.now().timestamp())
    last_inserted_data = list(g.db.data_collection.find().sort([('_id',-1)]).
    limit(1))
    if len(last_inserted_data) != 0:
        data['time_diff'] = float(data['time_insert'])-float(
    last_inserted_data[0]['time_insert'])
    else:
        data['time_diff'] = 0
    try:
        g.db.data_collection.insert_one(data)
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200


@api.route('/list/client', methods=['GET'])
def client_list():
    amount = request.args.get('latest', default=0, type=int)

    response = list(g.db.client_collection.find(ArgsValidation().client_args(
                                                request.args)).sort(
                                                "$natural", -1).limit(amount)
    )

    for document in response:
        del document['_id']

    return jsonify(response), 200

# ---------- Se  o de listagem de clientes contendo mais informa  es
    como last-update-time and number of services----------#
@api.route('/list/client_full', methods=['GET'])
def client_list_full():
    amount = request.args.get('latest', default=0, type=int)
    pipeline = [

        { '$lookup':
                    {
                        'from' : 'datas',
                        'localField' : 'chipset',
                        'foreignField' : 'chipset',
```

```
                        'as' : 'data-aggregation'
                    }
        }
    ]
    table_agregation = list(g.db.client_collection.aggregate(pipeline))

    for document in table_agregation:
        last_update_date = ""
        if(document["data-aggregation"] != []):
            last_update_date = document["data-aggregation"][-1]["_id"].
    generation_time.strftime("%d/%m/%y %H:%M:%S")

        count_services = g.db.service_collection.count({"mac": document["mac"
    ], "chipset": document["chipset"]})
        document['last-update-data'] = last_update_date
        document['number-of-services'] = count_services
        del document['_id']
        del document['data-aggregation']


    return jsonify(table_agregation), 200
# ---------- Finaliza  o de listagem contendo mais informa  es
    ---------- #

# ---------- Se  o de cria  o de regras provis rio ----------#
@api.route('/list/service')
def service_list():
    amount = request.args.get('latest', default=0, type=int)

    response = service_aggregate()

    return jsonify(response), 200


@api.route('/list/data')
def data_list():
    amount = request.args.get('latest', default=0, type=int)

    response = list(g.db.data_collection.find(ArgsValidation().data_args(
                                              request.args)).sort(
                                              "$natural", -1).limit(amount))


    for document in response:
        del document['_id']

    return jsonify(response), 200


```

```
@api.route('/report', methods=['GET'])
def report_data():
    LogReport().init_report_config()
    return jsonify({'code': 200, 'message': 'Success'}), 200


# ---------- Se  o de cria  o de regras provis rio ----------#

@api.route("/rule", methods=['POST'])
def insert_rule():
    rule = request.get_json() if request.is_json else request.form.to_dict()

    try:
        g.db.rule_collection.insert_one(rule)
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200

@api.route('/list/rule', methods=['GET'])
def list_rule():

    try:
        response = list(g.db.rule_collection.find(ArgsValidation.rule_args(
                                                  request.args )))
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500

    for document in response:
        del document['_id']

    return jsonify(response), 200

# ---------- Finaliza  o de cria  o de regras ---------- #



# ---------- List Interfaces (API para tela inicial de todos os
    projetos / gateway local) ---------- #

@api.route('/list/interface', methods=['GET'])
def list_interface():
    try:
        # ---------- Contagem de quantidade de clientes ----------
    #

        channel_list = ['HTTP','ZIGBEE','TCP','UDP']
        list_response = []
```

```
238         # Aggregation que calcula a quantidade de clientes que usam os
    protocolos especificados na lista channel_list
            for channel in channel_list:
240             json_reponse = {}
                json_reponse["name"] = channel
242             json_reponse["clients"] = 0
                json_reponse["services"] = 0
244             json_reponse["data_count"] = 0
                pipeline_clients_services = [
246                 {

248                     '$match':
                        {
250                         'channel':channel
                        }
252                 }
                    ,
254                 {
                        '$count':"count"
256                 }
                ]
258
                pipeline_data = [
260                 {
                        '$match':
262                     {
                            'channel':channel
264                     }
                    },
266                 {
                        '$group':
268                     {
                            '_id':'$channel',
270                         'channel_bytes_sum':{'$sum':'$size'}
                        }
272                 }
                ]
274
                client_list = list(g.db.client_collection.aggregate(
    pipeline_clients_services))
276             if len(client_list) != 0:
                    temp_channel = channel.lower()+"_clients"
278                 json_reponse["clients"] = client_list[0]['count']

280         # Aggregation que calcula a quantidade de servi os por mecanismo de
    comuni  o utilizado na lista lista
            service_list = list(g.db.service_collection.aggregate(
    pipeline_clients_services))
282         if len(service_list) != 0:
```

```
                temp_channel = channel.lower()+"_service"
284             json_reponse["services"] = service_list[0]['count']


286
        # Aggregation que calcula a quantidade de dados que usam o mecanismo
    de comunica  o utilizado na lista
288         data_list = list(g.db.data_collection.aggregate(pipeline_data))
            if len(data_list) != 0:
290             temp_channel = channel.lower()+"_data"
                json_reponse["data_count"] = data_list[0]['channel_bytes_sum'
    ]
292
            list_response.append(json_reponse)
294
     except errors.WriteError:
296         return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500

298  return jsonify(list_response), 200


300
# ------------------------------ Fim do list interface
    ------------------------------ #
302


304 # ------------------------------ List Gateway (listar informa  es acerca de
    gateways ) ------------------------------ #

306 @api.route('/list/gateway',methods=['GET'])
def list_gateway():
308     # Inserir a coleta dos dados associados ao gateway por meio da API que a
    Daiane est  fazendo

310     # Realiza a contagem de clientes/servi os/rules
     client_count = g.db.client_collection.count()
312  service_count = g.db.service_collection.count()
     rule_count = g.db.rule_collection.count()
314
     time_now = ObjectId.from_datetime(datetime.datetime.now()+datetime.
    timedelta(minutes=180))
316  time_now_minus_5 = ObjectId.from_datetime(datetime.datetime.now()+
    datetime.timedelta(minutes=180)-datetime.timedelta(minutes=5))

318  compressed_data_count = list(g.db.data_collection.aggregate([
         {
320          "$match":
             {
322              'type':'compressed',
                  '_id':{'$gt':time_now_minus_5}
```

```
            }
        }
        ,
        {
            '$group':
            {
                '_id':'$type',
                'count':{'$sum':1},
                'sum_size_compressed':{'$sum':'$size_compressed'},
                'sum_size_document':{'$sum':'$size'}
            }
        }

        ]))

    # Aggregate que soma o tamanho dos tipos compressed e uncompressed e
    retorna em chaves de nome diferente
    rate_sum = list(g.db.data_collection.aggregate([
        {
            "$match":
            {
                '_id':{'$gt':time_now_minus_5}
            }
        }
        ,
        {
            '$project':
            {
                '_id':'compressed',
                'compressed_sum':{
                    '$cond':{'if':{'$eq':['$type','compressed']},'then':{'
    $sum':'$size_compressed'},'else':{'$sum':0}}
                },
                'uncompressed_sum':{
                    '$cond':{'if':{'$eq':['$type','uncompressed']},'then':{'
    $sum':'$size'},'else':{'$sum':0}}
                }
            }
        }
        ,
        {
            '$group':
            {
                '_id':'compressed',
                'compressed_sum':{'$sum':'$compressed_sum'},
                'uncompressed_sum':{'$sum':'$uncompressed_sum'}
            }
        }
        ]))
```

```
370
372     # Aggregate que pega o documento mais recente da data_collection
        collections = [g.db.data_collection, g.db.service_collection, g.db.
    client_collection]
374     time_dict = {}
        for collection in collections:
376         latest_time = list(collection.aggregate([
                {
378                 '$sort':{'_id': -1}
                },
380             {
                    '$limit': 1
382             }
                ,
384             {
                    '$project':{
386                     '_id':'$_id'
                    }
388             }
            ]))
390         #print(latest_time[0]['_id'].generation_time)
            #print(latest_time[0]['_id'])
392         if len(latest_time) != 0:
                iso_timestamp = latest_time[0]['_id'].generation_time
394             time_value = time.mktime(iso_timestamp.timetuple())
                if(collection == g.db.data_collection):
396                 time_dict['data'] = time_value
                    time_dict['data_utc'] = iso_timestamp
398             elif(collection == g.db.service_collection):
                    time_dict['service'] = time_value
400                 time_dict['service_utc'] = iso_timestamp
                elif(collection == g.db.client_collection):
402                 time_dict['client'] = time_value
                    time_dict['client_utc'] = iso_timestamp
404         else:
                time_dict['data'] = 0
406             time_dict['service'] = 0
                time_dict['client'] = 0
408
        if time_dict['data'] == 0 and time_dict['service'] == 0 and time_dict['
    client'] == 0:
410         final_time = time_dict['data_utc']

412     type_list = ['client','service','data']
        biggest_time = 0
414     biggest_time_utc = ""
        for typeof in type_list:
416         if (time_dict[typeof] > biggest_time):
```

```
            biggest_time = time_dict[typeof]
418         biggest_time_utc = time_dict[typeof+"_utc"]

420     if len(rate_sum) != 0:
            compressed_sum = rate_sum[0]['compressed_sum']
422         uncompressed_sum = rate_sum[0]['uncompressed_sum']

424         compressed_rate = compressed_sum/300
            uncompressed_rate = uncompressed_sum/300
426     else:
            compressed_rate = 0
428         uncompressed_rate = 0



432     if len(compressed_data_count) != 0:
            # Dados sobre o batch de dados enviados, compactados ou n o
434         count = int(compressed_data_count[0]['count'])
            sum_size_compressed = int(compressed_data_count[0]['
    sum_size_compressed'])
436         sum_size_document = int(compressed_data_count[0]['sum_size_document'
    ])

438         # Calculando o percentual de utiliza   o do batch size
            compression_rate = (sum_size_compressed/sum_size_document)
440         percent = (1-compression_rate)*100
        else:
442         percent = 0

444     response = {
                'percent':round(percent,2),
446             'client_count':client_count,
                'service_count':service_count,
448             'rule_count':rule_count,
                'compacted_rate':round(compressed_rate,2),
450             'not_compacted_rate':round(uncompressed_rate,2),
                'last_updated':biggest_time_utc,
452             'gateway_name':0,
                'gateway_master':0
454     }
        response_list = []
456     response_list.append(response)

458     return jsonify(response_list),200



462 # ------------------------------ Fim de list gateway
    ---------------------------------- #
```

```
464 # ---------------------------- List service-cloud (listar informa  es acerca
        de service-cloud ) ---------------------------- #

466 @api.route('/list/service-cloud', methods=['GET'])
    def service_cloud():
468
        # Pega todos os clientes existentes na collection
470     amount = request.args.get('latest', default=0, type=int)
        clients = list(g.db.client_collection.find(ArgsValidation().client_args(
472                                                request.args)).sort(
                                                "$natural", -1).limit(amount)
        )
474
        service_list = service_aggregate()
476
        response_list = []
478     for client in clients:

480         for service in service_list:

482             response_json = {
                    'device_name':'None',
484                 'service_name':'None',
                    'value':0,
486                 'associate_rules':0,
                    'data_amount':0,
488                 'critical_messages':0,
                    'storage_time':0,
490                 'last_updated':'None'
                }
492
                if(service['mac'] == client['mac'] and service['chipset'] ==
    client['chipset']):
494
                    ## Aggregate para pegar todas as mensagens cr ticas
    associadas a um determinado servi o
496                 data_list = list(g.db.data_collection.aggregate([
                    {
498                     "$match":
                        {
500                         'sensitive':'1',
                            'chipset':service['chipset'],
502                         'mac':service['mac']
                        }
504                 }
                    ]))
506
                    ## Aggregate para pegar todas as mensagens associadas as um
```

```
    determinado servi o
508                 data_listed = list(g.db.data_collection.aggregate([
                    {
510                     "$match":
                        {
512                         'chipset':service['chipset'],
                            'mac':service['mac']
514                     }
                    }
516                 ]))

518                 ## Aggregate para pegar todas as regras associadas a um
    determinado servi o
                    rule_response = list(g.db.rule_collection.aggregate([
520                 {
                        "$match":
522                     {
                            'chipset':service['chipset'],
524                         'mac':service['mac']
                        }
526                 }
                    ,
528                 {
                        "$project":
530                     {
                            "_id":"rule",
532                         "count":{"$sum":1}
                        }
534                 }
                    ]))
536
                    # Coloca os valores adquiridos no dicion rio response_json
538                 response_json['device_name'] = client['name']
                    response_json['service_name'] = service['name']
540                 if len(rule_response) != 0:
                        response_json['associate_rules'] = rule_response[0]['
    count']
542                 response_json['data_amount'] = service['size']
                    response_json['critical_messages'] = len(data_list)
544                 response_json['last_updated'] = service['last-update-data']
                    try:
546                     response_json['value'] = str(data_listed[-1]['value'
    ][-1])
                    except IndexError:
548                     response_json['value'] = 'None'

550                 # Coloca o dicion rio na lista final de resposta
                    response_list.append(response_json)
552
```

```
    return jsonify(response_list),200

# ---------- Fim do list service-cloud ---------- #


# ---------- Listagem service-cloud-interface ---------- #
@api.route('/list/service-cloud-interface',methods=['GET'])
def service_cloud_interface():

    # Pega todos os clientes existentes na collection
    amount = request.args.get('latest', default=0, type=int)
    clients = list(g.db.client_collection.find(ArgsValidation().client_args(
                                                request.args)).sort(
                                                "$natural", -1).limit(amount)
    )


    response_list = []
    for client in clients:

        ## JSON que mostra o formato da resposta do m todo da API
        response_json = {
            'device_name':'None',
            'device_interface':'None',
            'device_identifier':0,
            'service_count':0,
            'last_updated':'None',
            'register_expiration':'None'
        }

        ## Aggregate que pega todos os dados associados ao cliente da
    itera   o atual
        data_response = list(g.db.data_collection.aggregate([
                {
                    "$match":
                    {
                        'chipset':client['chipset'],
                        'mac':client['mac']
                    }
                }
        ]))
        ## Pega o  ltimo  elemento da lista retornada e coloca em
    last_data_update o hor rio em que
        ## o documento foi inserido
        if(len(data_response) != 0):
            last_data_update = data_response[-1]['_id'].generation_time.
    strftime("%a - %d %b %Y %H:%M:%S")
        service_count = g.db.service_collection.count()
        response_json['device_name'] = client['name']
```

```
598         response_json['device_interface'] = client['channel']
            response_json['device_identifier'] = client['mac']
600         response_json['service_count'] = service_count
            response_json['last_updated'] = last_data_update
602
            response_list.append(response_json)
604
    return jsonify(response_list),200
606
# ------------------ FIM Listagem service-cloud-interface ---------------- #
608
# ------------------ Listagem service-cloud-average
    ------------------------- #
610 @api.route('/list/service-cloud-average',methods=['GET'])
def service_cloud_average():
612
    # Pega todos os clientes existentes na collection
614     amount = request.args.get('latest', default=0, type=int)
    clients = list(g.db.client_collection.find(ArgsValidation().client_args(
616                                                request.args)).sort(
                                                "$natural", -1).limit(amount)
    )
618
    service_list = service_aggregate()
620
    result_list = []
622     for client in clients:

624         for service in service_list:

626             # JSON que contem o formato da resposta do m todo
                response_json = {
628                 'device_name':'None',
                    'service_name':'None',
630                 'input_average':0,
                    'last_updated':'None'
632             }

634             # Aggregate que pega todos os dados associados a um servi o e
    calcula a m dia de time_diff
                data_time_average = list(g.db.data_collection.aggregate([
636                 {
                        "$match":
638                     {
                            'serviceNumber':service['number']
640                     }
                    }
642                 ,
                    {
```

```
                    "$group":
                    {
                        '_id':'average',
                        'average_input_time':{'$avg':'$time_diff'}
                    }
                }
            ]))


            if(service['mac'] == client['mac'] and service['chipset'] ==
    client['chipset']):
                response_json['device_name'] = client['name']
                response_json['service_name'] = service['name']
                response_json['last_updated'] = service['last-update-data']
                if len(data_time_average) != 0:
                    response_json['input_average'] = data_time_average[0]['
    average_input_time']

                result_list.append(response_json)

    return jsonify(result_list),200

# ---------------------------------------- Metodo de inser  o de dados
    enviados ---------------------- #
@api.route('/telemetry',methods=['POST'])
def send_data_telemetry():
    telemetry_dict = request.get_json() if request.is_json else request.form.
    to_dict()

    try:
        g.db.telemetry_collection.insert_one(telemetry_dict)
    except errors.WriteError:
        return jsonify({'code': 500, 'message': 'Internal Server Error'}),
    500
    return jsonify({'code': 200, 'message': 'Success'}), 200

# ---------------------------------- FIM do metodo de inser  o de dados
    enviados ---------------------- #


# ---------------------------------------- Metodo de inser  o de dados
    enviados ---------------------- #
@api.route('/telemetry-gateway',methods=['POST'])
def send_data_telemetry_gateway():
    telemetry_dict = request.get_json() if request.is_json else request.form.
    to_dict()
    gateway = list(g.db.telemetry_collection.find_one({'remote_gateway_name':
     telemetry_dict['remote_gateway_name']}))  if g.db.telemetry_collection.
    find_one({'remote_gateway_name': telemetry_dict['remote_gateway_name']})
```

```
    != None else []
    if len(gateway) != 0:
684         try:
                g.db.telemetry_collection.insert_one(telemetry_dict)
686         except errors.WriteError:
                return jsonify({'code': 500, 'message': 'Internal Server Error'})
    , 500
688         return jsonify({'code': 200, 'message': 'Success'}), 200
    else:
690         return jsonify({'code':400,'message':'remote-gateway already exists'
    }),400

692 # ---------------------------------- FIM do m todo de inser   o de dados
    enviados ---------------------- #

694 @api.route('/list/gateway-cloud',methods=['GET'])
def list_gateway_cloud():
696
    # Gera o ObjectID de agora e 5 minutos atr s
698     time_now = ObjectId.from_datetime(datetime.datetime.now()+datetime.
    timedelta(minutes=180))
    time_now_minus_5 = ObjectId.from_datetime(datetime.datetime.now()+
    datetime.timedelta(minutes=180)-datetime.timedelta(minutes=5))
700
    # Pega todos os remote gateways cadastrados em telemetry_collection
702     remote_gateways = list(g.db.telemetry_collection.find({'
    remote_gateway_name':{'$exists':'true'}}))

704     print("REMOTE GATEWAYS: {}".format(remote_gateways))

706     response_list = []
    for remote_gateway in remote_gateways:
708
        response = {
710             'remote_gateway':'None',
            'total_sent_compacted':0,
712             'economy':0,
            'sensitive_transmission':0,
714             'compacted_transmission':0
        }
716
        uncompressed_data = list(g.db.telemetry_collection.aggregate([
718         {
            "$match":
720             {
                'type':'uncompressed',
722                 '_id':{'$gt':time_now_minus_5},
                'remote_gateway':remote_gateway['remote_gateway_name']
724             }
```

```
        }
        ,
        {
            '$group':
            {
                '_id':'$type',
                'count':{'$sum':1},
                'sum_size':{'$sum':'$sent_size'},
            }
        }

        ]))

        compressed_data = list(g.db.telemetry_collection.aggregate([
        {
             "$match":
             {
                  'type':'compressed',
                   '_id':{'$gt':time_now_minus_5},
                   'remote_gateway':remote_gateway['remote_gateway_name']
             }
        }
        ,
        {
            '$group':
            {
                '_id':'$type',
                'count':{'$sum':1},
                'sum_size':{'$sum':'$sent_size'},
            }
        }
        ]))

        sensitive_transmission = list(g.db.telemetry_collection.aggregate([
        {
             "$match":
             {
                  'senstive':1,
                   '_id':{'$gt':time_now_minus_5},
                   'remote_gateway':remote_gateway['remote_gateway_name']
             }
        }
        ,
        {
            '$group':
            {
                '_id':'$type',
                'count':{'$sum':1},
                'sum_size':{'$sum':'$sent_size'},
```

```
            }
        }
        ]))

        if len(compressed_data) != 0:
            response['compacted_transmission'] = compressed_data[0]['sum_size
    ']/300
            response['total_sent_compacted'] = compressed_data[0]['sum_size']

        if len(sensitive_transmission) != 0:
            response['sensitive_transmission'] = sensitive_transmission[0]['
    sum_size']/300

        response['remote_gateway'] = remote_gateway['remote_gateway_name']

        response_list.append(response)

    return jsonify(response_list),200
```

code/app.py

main.py

```
from dims import create_app
import subprocess
import socket
import click
import os

app = create_app()


@click.group(invoke_without_command=True)
@click.option('--root', is_flag=True,
              help='Flag that indicates if not is a root.')
@click.option('-p', '--peerport', type=int, default=5010, help='Peer port.')
@click.option('-r', '--rootport', type=int, default=5010, help='Root port.')
def start_server(root, peerport, rootport):
    # starts chord
    local_ip = socket.gethostbyname(socket.getfqdn())
    args = f'-m 1 -p {peerport} -h {local_ip}'.split(' ') if root else \
    f'-m 0 -p {peerport} -h {local_ip} -r {rootport} -R {local_ip}'.split(' '
    )
    argv = ['dht_peer', *args]
    subprocess.Popen(argv)

    # dims flask
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port)
    return True
```

```
28
if __name__ == '__main__':
30     start_server()
```

code/__main__.py