SINCRONIZAÇÃO DE TEMPO E ALOCAÇÃO DE RECURSOS EM SISTEMAS DA INTERNET DAS COISAS

Tiago Cariolano de Souza Xavier

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Cláudio Luis de Amorim
Flávia Coimbra Delicato

Rio de Janeiro
Junho de 2020

SINCRONIZAÇÃO DE TEMPO E ALOCAÇÃO DE RECURSOS EM SISTEMAS DA INTERNET DAS COISAS

Tiago Cariolano de Souza Xavier

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Cláudio Luis de Amorim
Flávia Coimbra Delicato

Aprovada por: Prof. Cláudio Luis de Amorim, D.Sc.
Profª. Flávia Coimbra Delicato, D.Sc.
Prof. Felipe Maia Galvão França, D.Sc
Profª. Noemi de La Rocque Rodriguez, D.Sc.
Prof. Célio Vinicius Neves de Albuquerque, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 2020

Xavier, Tiago Cariolano de Souza
Sincronização de Tempo e Alocação de Recursos em Sistemas da Internet das Coisas / Tiago Cariolano de Souza Xavier. – Rio de Janeiro: UFRJ/COPPE, 2020.
XI, 97 p.: il.; 29,7 cm.
Orientadores: Cláudio Luis de Amorim
Flávia Coimbra Delicato
Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2020.
Referências Bibliográficas: p. 91-97.
1. Sincronização de Tempo. 2. Alocação de Recursos. 3. Internet das Coisas. I. Amorim, Cláudio Luis de *et al*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico este trabalho aos meus pais, Izaura de Souza Xavier e Joaquim Cariolano Xavier, e a minha querida, já falecida, avó materna Maria Rodriguez dos Santos, os quais são minhas fontes de valores e dedicaram muito esforço e sacrífico para fornecer a melhor educação que eu pude alcançar.

Agradecimentos

Agradeço primeiramente à Deus, Nosso Senhor Jesus Cristo, pois sem Ele nada seria possível.

Agradeço com todo meu amor aos meus pais, Izaura e Joaquim, que sempre me apoiaram em todos os momentos e foram a razão para eu ter ido até o fim desse trabalho.

Agradeço a minha namorada, Emanuelle, que sempre esteve por perto com seu jeito doce quando eu precisei.

Agradeço ao professor Cláudio pela paciência, compreensão e a oportunidade de trabalhar e aprender no seu laboratório.

Agradeço à professora Flávia pela atenção despendida. A quem passei a admirar por sua capacidade e paixão pelo o que faz, e que ajudou-me de maneira fundamental para que eu pudesse concluir esse trabalho.

Por fim, agradeço a todos os familiares, amigos, colegas, professores e funcionários que de alguma forma me auxiliaram no decorrer desta jornada.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SINCRONIZAÇÃO DE TEMPO E ALOCAÇÃO DE RECURSOS EM SISTEMAS DA INTERNET DAS COISAS

Tiago Cariolano de Souza Xavier

Junho/2020

Orientadores: Cláudio Luis de Amorim
Flávia Coimbra Delicato

Programa: Engenharia de Sistemas e Computação

O paradigma da Internet das Coisas (IoT - Internet of Things) consiste da interconexão dinâmica de objetos autônomos a fim de estender a Internet para o mundo físico. Para atender os requisitos das aplicações, arquiteturas da IoT separam o sistema em três níveis: nível das coisas, nível da borda e nível da nuvem. Cada nível deve fornecer serviços de infraestrutura que ajudam a alavancar o desempenho das aplicações ao mesmo que devem ser eficientes. Nesta tese, dois problemas da infraestrutura de sistemas da IoT são abordados: a sincronização de tempo no nível das coisas e a alocação de recursos no nível da borda. A sincronização de tempo é o serviço de infraestrutura que visa manter todos os nós com uma visão comum o tempo. A escolha do nó de referência, aquele que fornece o relógio de referência, é uma estratégia que pode mitigar o erro de sincronização entre os nós da rede. Nesta tese, foi proposto um algoritmo distribuído de sincronização de tempo eficiente energeticamente que visa selecionar o nó que minimiza sua distância para os demais nós da rede. Esse algoritmo orquestra a configuração de temporizadores nos nós sensores, gerando o mecanismo de seleção do nó de referência que não necessita do envio adicional de mensagens de sincronização. A alocação de recursos no nível de borda visa escolher os recursos físicos e virtuais disponíveis para uso das aplicações. Nesta tese, é proposto um novo algoritmo distribuído de alocação de recursos que suporta a heterogeneidade das aplicações e dos dispositivos da camada das coisas. O alocador de recursos explora a natureza descentralizada do nível de borda, promovendo a colaboração no processo de alocação. Ele também fornece um eficiente uso de recursos ao mesmo tempo que atende os requisitos de latência, heterogeneidade e frescor de dado das aplicações da IoT, respeitando suas diferentes prioridades.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

TIME SYNCHRONIZATION AND RESOURCE MANAGEMENT IN INTERNET OF THINGS SYSTEMS

Tiago Cariolano de Souza Xavier

June/2020

Advisors: Cláudio Luis de Amorim
Flávia Coimbra Delicato

Department: Systems Engineering and Computers Science

The Internet of Things (IoT) paradigm consists of the dynamic interconnection of autonomous objects in order to extend the Internet to the physical world. To meet application requirements, IoT architectures separate the system in three tiers: things tier, edge tier and cloud tier. Each tier must provide infrastructure services that help to leverage the applications performance while being efficient. In this thesis, two problems of the IoT systems infrastructure are addressed: the time synchronization on things tier and the resource allocation in edge tier. Time synchronization is the infrastructure service that aims to keep all nodes with a common view of the time. The choice of the reference node, which provides the reference time, is a strategy that can mitigate the synchronization error between the network nodes. In this thesis, an energy-efficient distributed time synchronization algorithm was proposed that aims to select the node that minimizes its distance to the other nodes in the network. The algorithm orchestrates the configuration of timers on sensor nodes, generating a mechanism for triggering the selection of the reference node without additional synchronization message exchanges. The resource allocation at the edge tier aims to choose physical and virtual available resources for use by the applications. In this thesis, a new distributed resource allocation algorithm is proposed that supports the heterogeneity of applications and devices of the things tier. The resource allocator explores the decentralized nature of the edge tier, promoting collaboration in the allocation process. It also provides an efficient use of resources while meeting the latency, heterogeneity and data freshness requirements of IoT applications, respecting their different priorities.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O paradigma da Internet das Coisas (IoT - Internet of Things) pode ser definido sob diferentes perspectivas, mas em geral se refere à interconexão de sensores, autuadores e outros objetos inteligentes, estendendo as interações existentes entre homem e máquina providas pela Internet atual para uma nova dimensão da comunicação homem-objeto e objetos-objeto (DELICATO *et al.* (2017). Vários tipos de dispositivos com características heterogêneas compõem as fontes de dados da IoT. Tais dispositivos possuem recursos limitados (capacidade de processamento, memória e comunicação), de modo que o processamento de tarefas e o armazenamento permanente de dados gerados, em geral, não podem ser executados pelos próprios dispositivos.

Plataformas de computação na nuvem possuem recursos virtualmente ilimitados em termos de capacidade de armazenamento, poder de processamento e comunicação, sendo tipicamente adotadas como backend em sistemas da IoT. O paradigma emergente que preconiza a interação de uma camada de dispositivos da IoT (fonte de dados) e a camada de nuvem é comumente denominado de Nuvem das Coisas (CoT - Cloud of Things) e tem sido investigado recentemente (CHANG *et al.* (2011), JIEHAN ZHOU *et al.* (2013), YOUSEFPOUR *et al.* (2018)). Neste paradigma, a nuvem age essencialmente como uma camada intermediária entre os objetos (coisas) inteligentes e as aplicações que fazem uso de dados e recursos fornecidos por esses objetos. Em tal contexto, sistemas da IoT se beneficiam dos poderosos recursos das plataformas de computação em nuvem para executar tarefas complexas e o gerenciamento de serviços provisionados por objetos inteligentes e os dados que eles produzem. Por outro lado, plataformas de computação na nuvem se beneficiam ampliando seu âmbito de operação para passar a lidar com objetos do mundo real.

Apesar dos benefícios da utilização de plataformas da nuvem como sistemas para processar requisições da IoT, a natureza essencialmente centralizada da nuvem não se adequa bem com a natureza inerentemente descentralizada da IoT. Enviar os dados dos dispositivos da IoT para processamento e armazenamento centralizado na

nuvem e, então, encaminhar tais dados para os usuários, pode resultar em atrasos intoleráveis para algumas aplicações. Para algumas aplicações, o tempo de resposta é um requisito crítico e a latência e a imprevisibilidade da comunicação com a nuvem pode levar a degradação de desempenho do sistema. Além disso, muitos dispositivos da IoT possuem capacidade de processamento que podem ser aproveitadas pelas aplicações. Os recursos desses dispositivos da IoT, bem como de gateways e outros nós na borda da rede podem ser explorados para executar processamento distribuído e próximo à fonte de dados.

O suporte para processamento de dados descentralizado nos dispositivos próximos à borda da rede, em combinação com os benefícios de tecnologias da nuvem, têm sido explorados no paradigma de Computação de Borda (Edge Computing) LI *et al.* (2017), LENKA *et al.* (2018), VASCONCELOS *et al.* (2019)). Tal paradigma tem como objetivo reduzir a sobrecarga (overhead) de comunicação e o tempo de transferência de dados e assim, reduzir a latência para as aplicações. A combinação de dispositivos da IoT (tais como nós sensores, smartphones e objetos inteligentes), nós de borda (tais como servidores de borda ou roteadores inteligentes) e plataformas da nuvem leva ao surgimento de uma arquitetura de três camadas - camada das coisas, camada da borda e camada da nuvem - para o sistema da CoT, com dispositivos de cada camada encarregados de diferentes responsabilidades e fornecendo diferentes tipos de serviços.

Os recursos disponíveis nos dispositivos da IoT, nos dispositivos de borda e na nuvem, bem como os serviços fornecidos em cada uma das camadas, compõem a infraestrutura do sistema da CoT. Os serviços de infraestrutura da CoT são aqueles que dão suporte para a execução do sistema e podem atuar em cada uma das camadas isoladamente, ou podem cruzar duas ou mesmo três camadas. Neste trabalho, nós estamos preocupados com dois serviços de infraestrutura que geram impacto no desempenho do atendimento das requisições de aplicações emergentes da IoT. O primeiro serviço, o qual é denominado sincronização de tempo, é executado na camada das coisas e consiste em manter os relógios dos nós com uma visão comum do tempo. A sincronização de tempo é um serviço de infraestrutura requerido por aplicações que demandam a ordenação temporal de eventos (coleta de dados ou interação entre dispositivos da IoT, por exemplo).

O segundo serviço de infraestrutura é a alocação de recursos para atendimento das aplicações da CoT. A alocação de recursos consiste em distribuir os recursos físicos e computacionais da CoT para atender as demandas das aplicações. Algumas propostas para alocação de recursos da CoT assumem uma arquitetura de três camadas, como a que mencionamos anteriormente. Tais propostas são as mais heterogêneas e complexas em termos de entidades computacionais envolvidas, e abrem várias possibilidades sobre como gerenciar com eficiência os recursos disponíveis DE-

LICATO *et al.* (2017). Algumas abordagens consideram a presença apenas da camada das coisas e da nuvem. Ainda, algumas estratégias assumem apenas a camada das coisas e um ou mais gateways, os quais possuem vários níveis de complexidade e responsabilidades em termos de gerenciamento de recursos. Finalmente, alguns trabalhos consideram apenas os recursos fornecidos pelas coisas e como gerenciar e atender as requisições das aplicações eficientemente. Nossa proposta de alocação de recursos considera a presença de três camadas e os recursos são alocados na camada da borda ou na nuvem.

1.1 Motivações

Nesta seção nós apresentamos as motivações que levaram a investigação do problema da sincronização de tempo na camada das coisas, bem como do problema de alocação de recursos na camada de borda.

1.1.1 O Problema de Sincronização de Tempo

Na arquitetura da CoT, a camada das coisas é composta por entidades físicas, as quais correspondem a qualquer objeto físico de interesse para fins de sensoriamento ou atuação. As entidades físicas integram recursos de processamento e comunicação (sem fio) e são instrumentadas com sensores ou atuadores, ao que é dado o nome de dispositivo da IoT ou dispositivo fim (end device). Os dados do ambiente físico são capturados por meio de uma rede de sensoriamento. Dentre os diversos tipos de redes de sensoriamento que podem compor a camada das coisas, a Rede de Sensores Sem Fio (RSSF) compõe uma grande parte dos domínios de aplicações da camada das coisas da CoT e é como uma ponte do mundo físico para o mundo digital, conforme descrito por LENKA *et al.* (2018). A RSSF é uma rede composta por um grupo de nós sensores espacialmente dispersos, dedicados em observar os fenômenos do ambiente físico, e que coletam e organizam os dados para um local central.

Comparados com dispositivos de outros sistemas distribuídos, ou mesmo com vários dispositivos da IoT, como smartphones, os nós da RSSF possuem grandes restrições de recursos. Tais restrições das RSSF trazem desafios para o projeto dos algoritmos que gerenciam as atividades e as tarefas dos nós sensores. Entre as restrições essenciais que o projeto da RSSF deve abordar, o uso eficiente de energia é um dos principais fatores a serem considerados (QIU *et al.* (2015)). Dessa forma, os serviços de infraestrutura fornecidos pela RSSF devem atender as demandas dos clientes ao mesmo tempo em que são eficientes energeticamente.

Um serviço de infraestrutura essencial para as RSSFs corresponde à manutenção de uma visão comum de tempo entre todos os nós sensores. Diversos domínios de

aplicações da IoT que usam a RSSF como rede de sensoriamento, tais como monitoramento de ambiente, gerenciamento de energia e detecção de perigo, demandam que os nós sensores possuam relógios sincronizados. São aplicações em que se exige manter a consistência de dados e a coordenação dos nós sensores ao mesmo tempo em que se respeita os limites temporais especificados.

Para que a sincronização temporal dos nós sensores seja mantida pela RSSF, um protocolo de sincronização de tempo deve ser empregado em todos os nós do sistema distribuído. O protocolo de sincronização de tempo mantém a diferença de tempo entre os nós sensores e um nó de referência restringida a um limite especificado, denominado de erro de sincronização.

Embora diversos protocolos de sincronização de tempo sejam capazes de fornecer uma noção comum de tempo com alta acurácia para toda a rede, o principal desafio com relação a esse serviço de infraestrutura consiste em alavancar o ganho de acurácia sem aumentar o número de mensagens trocadas, bem como o consumo de energia dos nós sensores. Além disso, o aumento da escala da rede traz complexidade ao desafio de sincronização de tempo, pois faz com que a abordagem de sincronização comece a incorrer no problema do erro cumulativo: a perda de acurácia durante a manutenção do estado de sincronização, devido a grandes distâncias em hops (saltos) entre os nós sensores e o nó de referência da rede (LENZEN *et al.* (2008), LI *et al.* (2014)).

O problema do erro cumulativo pode ser abordado por meio de uma seleção adequada do nó de referência, porém, devido ao custo necessário para selecionar o nó de referência com o objetivo de impedir o aumento da distância dele para os demais nós sensores para, essa abordagem não foi tratada suficientemente em trabalhos recentes da literatura de RSSF. Nem em algoritmos focados em acurácia como o proposto por JIA *et al.* (2019), nem em algoritmos que visam eficiência energética como a abordagem de QIU *et al.* (2018), por exemplo.

1.1.2 O Problema de Alocação de Recursos

A computação na nuvem tornou-se uma plataforma atraente para uma variedade de aplicações da IoT, por causa das suas propriedades de computação elásticas, sua enorme disponibilidade de recursos e sua razão custo-benefício. No entanto, os rigorosos requisitos de qualidade de serviço (Quality of Service - QoS) de algumas aplicações da IoT exigem desempenho previsível da nuvem e baixas latências fim-a-fim entre o usuário e a nuvem (SHEKHAR e GOKHALE (2017)).

Os recursos disponíveis na nuvem, embora abundantes, nem sempre são capazes de atender às restrições de latência de aplicações sensitivas à latência, devido à grande distância entre um datacenter de uma plataforma na nuvem e os dispositivos

que originam os dados. Além disso, outro problema que impede o atendimento das restrições de aplicações críticas se refere a grande distância entre a nuvem e o dispositivo que executa a aplicação que requisita o dado. A adoção de computação na borda da rede pode auxiliar a nuvem a fornecer serviços que atendam a tais requisitos temporais. A Computação de Borda visa fornecer os benefícios da computação em nuvem sem incorrer em seus problemas como, por exemplo, a alta latência e o consumo de banda do núcleo da rede. A Computação de Borda também traz infraestrutura virtualizada e provisionamento de recursos elásticos sob demanda para mais perto dos dispositivos finais, na borda da rede.

A nuvem e a borda são fortemente baseados em virtualização como descrito por LI *et al.* (2017). O processo de virtualização mapeia um ou mais recursos físicos para um ou mais recursos virtuais para serem fornecidos como serviços para as aplicações. Considerando a limitação de recursos dos dispositivos de borda, em comparação com os datacenters na nuvem, a virtualização dos recursos deve ser realizada sem aumentar o custo operacional do sistema. Dessa forma, propostas recentes propõem modelos de virtualização em que um conjunto de nós virtuais elásticos (por exemplo, containers) é feito disponível para aplicações, fornecendo um desacoplamento entre a infraestrutura física e as aplicações. Atribuir tais nós virtuais para as requisições de aplicações de uma maneira oportuna e eficiente faz despontar o problema da alocação de recursos na camada de borda.

Similar ao tradicional gerenciamento de recursos na nuvem, o objetivo do gerenciamento de recursos na camada de borda é maximizar o número de requisitos de aplicações atendidos pela infraestrutura, enquanto garantindo o custo operacional alvo. No entanto, devido aos requisitos específicos de aplicações da IoT e a heterogeneidade dos dispositivos, a alocação de recursos requer soluções não-triviais e desenvolvidas sob medida. Além disso, em sistemas da CoT sem a camada de borda da rede, a nuvem é um provedor de recursos centralizado com visão global dos recursos, enquanto em sistemas da CoT em que a computação na borda é adotada, os nós de borda, que são os provedores de recursos próximos à fonte de dados, estão espalhados geograficamente em uma rede distribuída.

1.2 Desafios de pesquisa

Nesta seção, nós descrevemos os desafios de pesquisa encontrados na literatura atual sobre sincronização de tempo na camada das coisas e gerenciamento de recursos na camada de borda da CoT.

1.2.1 O Desafio da Sincronização de Tempo na Camada das Coisas

Cada nó sensor possui um relógio composto de um oscilador de cristal que emite um sinal em uma frequência nominal. A frequência de operação do oscilador difere da frequência nominal e varia por causa das diferenças no processo de fabricação de cada circuito. Além disso, após a implantação (deployment) dos nós sensores no ambiente físico, os relógios dos nós sensores podem variar devido as condições do ambiente, como temperatura e umidade, as quais afetam o cristal do relógio. Cada nó sensor posicionado em uma área geográfica experimenta diferentes condições e o oscilador de cristal pode variar a frequência do sinal. O processo de sincronização de tempo dos relógios consiste em aproximar periodicamente os valores dos relógios de cada nó sensor para o valor de relógio de um nó de referência. A sincronização de tempo, como um serviço de infraestrutura da IoT, deve atender ao requisito de acurácia temporal de aplicações de mais alto nível. No entanto, a sincronização de tempo deve garantir tal acurácia usando a menor quantidade de recursos. O serviço de sincronização não deve causar uma sobrecarga na rede que impacte o desempenho das aplicações, bem como não deve consumir uma quantidade de energia que comprometa o tempo de vida útil dos nós sensores.

Protocolos como o NTP, proposto por MILLS (1991), o qual é usado para sincronização na Internet, não é adequado para RSSF, pois requer uma grande quantidade de recursos. Protocolos projetados exclusivamente para RSSFs como os descritos por MARÓTI *et al.* (2004), LENZEN *et al.* (2015) e LIM *et al.* (2016) foram propostos para alcançar alta acurácia de sincronização de tempo, contudo, eles não são eficientes energeticamente. Esses protocolos podem atingir alta acurácia, mas todos os nós sensores devem trocar mensagens de sincronização periodicamente. Devido ao número elevado de troca de mensagens e complexos cálculos, a alta acurácia é alcançada em sacrifício de um excessivo consumo de energia.

Os protocolos PBS e o GPA propostos por NOH e SERPEDIN (2007) e NOH *et al.* (2008) são esquemas de sincronização eficientes energeticamente, porque parte dos nós sensores é capaz de sincronizar seus relógios sem enviar nenhuma mensagem de sincronização. Esses nós corrigem seus relógios apenas sondando a troca de mensagens de seus vizinhos. O protocolo de sincronização STETS, desenvolvido por QIU *et al.* (2015) utiliza uma abordagem similar a do GPA, porém com um mecanismo de temporização eficiente para selecionar os nós que não enviam mensagens de sincronização. O problema, contudo, é que para diversos cenários de topologia de rede, alguns nós não são capazes de sincronizar seus relógios. O protocolo de QIU *et al.* (2018) melhora o STETS e é capaz de sincronizar a maioria dos nós da rede.

Os protocolos mencionados acima não tratam o problema do erro cumulativo.

Por exemplo, com relação à seleção do nó de referência da rede, protocolos tais como os propostos por MARÓTI *et al.* (2004), LENZEN *et al.* (2015) e LIM *et al.* (2016) selecionam o nó de referência com o menor número de identificador de nó da rede. Embora tal estratégia não requeira alto poder de processamento do nó sensor, ela não causa impacto na acurácia ou na economia de energia. Já protocolos que são eficientes energeticamente (NOH e SERPEDIN (2007), NOH *et al.* (2008), QIU *et al.* (2015)) selecionam o nó de referência aleatoriamente. O RSync (QIU *et al.* (2018)) seleciona o nó de referência com a menor quantidade de energia consumida, porém esta estratégia não seleciona o nó de referência almejando mitigar o erro cumulativo o qual aumenta a acurácia do algoritmo.

Dessa maneira, este campo ainda permanece em aberto para que soluções eficientes possam tratar este problema. Neste trabalho, nós consideramos o desafio de selecionar o nó de referência da rede de maneira a mitigar o problema do erro cumulativo, e proporcionar um aumento da acurácia dos valores de relógio dos nós sensores sem o aumento do overhead de comunicação e do consumo energético.

1.2.2 O Desafio do Gerenciamento de Recursos na Nuvem das Coisas

Em um ambiente da CoT, as aplicações acessam a camada de borda da rede por meio do envio de requisições, as quais contém descrições de tarefas, que geram cargas de trabalho que devem ser executadas pela infraestrutura física. A fim de ocultar dos clientes os detalhes da infraestrutura, plataformas e dados disponíveis, permitindo o compartilhamento de recursos entre as aplicações, os nós da camada de borda fazem uso de virtualização. A virtualização consiste em disponibilizar nós virtuais para a execução e atendimento das requisições das aplicações, permitindo o desacoplamento entre a infraestrutura da CoT e as aplicações.

Uma vez virtualizados os serviços providos, surge um importante desafio de como distribuir as cargas de trabalho da aplicação nos dispositivos físicos da camada de borda que compõem a infraestrutura. Esta atividade corresponde à alocação de recursos dos nós envolvidos na execução dos serviços (de sensoriamento, atuação e outros) para os usuários e aplicações que fazem parte da CoT. Processos de alocação de recursos devem ser concebidos para tomar as melhores decisões, buscando atingir dois objetivos principais: atender ao máximo os requisitos das aplicações, e consumir o mínimo de recursos da infraestrutura de CoT.

A latência das aplicações é um dos principais requisitos da CoT, sendo o atendimento desse requisito uma das principais razões para a adoção de computação na borda da rede. Aplicações reais também são heterogêneas, o que significa que elas requisitam dados de dispositivos da IoT heterogêneos, com diferentes características

e requisitos. Além disso, em cenários reais da CoT, diferentes usuários possuem níveis de prioridades distintos, demandando um escalonamento do provisionamento dos dados. Finalmente, entre os possíveis requisitos de aplicações emergentes da CoT, o frescor do dado se destaca, devido a sua importância em sistemas baseados em sensoriamento de dados (SANTOS *et al.* (2019)).

A alocação de recursos deve garantir o provisionamento do dado solicitado, ao mesmo tempo que atende todos os requisitos das aplicações. Em sistemas da CoT de duas camadas a tarefa de alocação de recursos é facilitada, pois a nuvem tem a visão global da camada das coisas. No entanto, a grande distância entre a fonte de dados (dispositivos da IoT) e a nuvem traz grandes dificuldades para o alocador de recursos entregar o dado com a latência requirida e respeitando os todos os requisitos da aplicação.

A computação na borda da rede aproxima a fonte de dados das aplicações, no entanto ela traz novas complicações com relação a alocação de recursos. O compartilhamento do dado com diferentes aplicações se torna mais complexo, pois, ao invés de ocorrer em um datacenter centralizado na nuvem, ele ocorre em uma rede distribuída na borda da rede. Além disso, enquanto a nuvem é abundante de recursos, os dispositivos da borda são limitados em capacidade de processamento, memória e comunicação.

Dessa forma, a distribuição dos recursos físicos na camada de borda,com o objetivo de atender os requisitos da aplicação, conduz ao desafio do desenvolvimento de técnicas mais sofisticadas de alocação de recursos e soluções específicas para cada tipo de ambiente. Esse campo de pesquisas, o qual é dinâmico, requer novas soluções para os desafios apresentados acima.

1.3 Questões de pesquisa

Os desafios de pesquisa descritos na seção anterior dão substrato para as seguintes duas questões, as quais são respondidas por essa tese:

Questão 1: Como prover um processo de sincronização de tempo com alta acurácia para RSSFs, mitigando o problema do erro cumulativo, de maneira que a sobrecarga decorrente da comunicação e a energia gasta entre os nós sensores não sejam aumentados?

Questão 2: Como fornecer dados de sensoriamento para atender requisições das aplicações emergentes da IoT, por meio de um processo de alocação de recursos eficiente e ditribuído da infraestrutura da CoT, atendendo os requisitos de latência, heterogeneidade das requisições, prioridade e frescor do dado das aplicações, utilizando um modelo de virtualização que leve em conta a natureza heterogênea dos dispositivos da IoT, e fazendo uso de computação de borda de maneira transparente

ao usuário?

1.4 Objetivos da tese

A fim de fornecer soluções para os desafios mencionados na seção anterior e avançar o estado da arte com relação ao desenvolvimento de protocolos de sincronização de tempo para RSSFs e à alocação de recursos na camada de borda, os dois principais objetivos deste trabalho são descritos a seguir.

O primeiro objetivo consiste em propor um algoritmo distribuído para sincronização de tempo em RSSFs de um sistema da CoT, o qual seleciona o nó de referência da rede, com o propósito de mitigar o problema do erro cumulativo. Tal algoritmo deve atender os seguintes requisitos: (i) reduzir a distância em hops dos nós sensores para o nó de referência e assim aumentar a acurácia da sincronização de tempo; (ii) ser eficiente energeticamente, evitando a troca de mensagens entre os nós sensores.

O algoritmo proposto neste trabalho seleciona o nó de referência quase-ótimo da rede, o qual minimiza a distância dos nós sensores para o nó de referência. O esquema de sincronização proposto também utiliza as próprias mensagens de sincronização para selecionar o nó de referência da rede, evitando troca de mensagens entre os nós sensores. O algoritmo usa um mecanismo de temporizadores nos nós sensores para selecionar o nó de referência sem a necessidade de propagar a informação da topologia da rede.

O segundo objetivo desta tese consiste em propor um novo algoritmo de alocação de recursos para um sistema da CoT de três camadas que (i) suporta a heterogeneidade dos dispositivos e aplicações, (ii) alavanca a natureza distribuída dos nós de borda para promover colaboração durante o processo de alocação e (iii) fornece um uso eficiente dos recursos do sistema enquanto atende os requisitos de latência e considera as diferentes prioridades das aplicações da IoT.

O algoritmo de alocação de recursos para a camada de borda da CoT proposto nesta tese visa atender este objetivo. Para compensar a menor quantidade de recursos dos nós de borda (em comparação com datacenters na nuvem), o algoritmo divide a carga de trabalho entre um conjunto de dispositivos de borda, os quais cooperam para alcançar os objetivos desejados. A latência é considerada como o principal requisito de QoS a ser atendido pelo processo de alocação de recursos. O frescor do dado é outro requisito considerado pelo algoritmo.

Por último, nosso algoritmo é capaz de acomodar diferentes prioridades de aplicação. Certamente, uma aplicação que monitora os sinais vitais de um paciente, ou uma aplicação que realiza a detecção de fogo, é mais crítica do que uma aplicação que apenas melhora o conforto dos usuários. Assim, nossa solução para a

alocação de recursos na CoT fornece mecanismos para representar a prioridade das aplicações e garantir que aquelas prioridades sejam respeitadas durante o processo de alocação de recursos.

Na próxima seção, nós apresentamos as contribuições da tese no avanço das áreas relacionadas aos desafios expostos acima.

1.5 Contribuições da tese

Considerando o problema de sincronização de tempo em RSSFs, nosso trabalho apresenta as seguintes contribuições. Nosso algoritmo mantém a atividade de sincronização de tempo utilizando uma abordagem, a qual é energeticamente eficiente. Nós propomos um algoritmo que visa selecionar o nó de referência ótimo da rede e que minimiza a distância dele para todos os demais nós da rede. O algoritmo utiliza as próprias mensagens de sincronização para selecionar o nó de referência, evitando custos extras de comunicação. O algoritmo melhora a acurácia da sincronização, em comparação com protocolos que representam o estado-da-arte sobre o tema, porém não aumenta o número de mensagens trocadas pelos nós sensores.

Com relação ao problema de alocação de recursos na camada de borda, nosso trabalho apresenta as seguintes contribuições, as quais são corroboradas pelas avaliações executadas. Primeiro, nosso algoritmo de alocação de recursos reduz o tempo de resposta para as aplicações e o consumo de energia dos dispositivos da IoT, em comparação com uma abordagem de duas camadas baseada apenas no uso da nuvem. Segundo, o tráfico de rede entre os nós, e entre as camadas de borda e nuvem, é consideravelmente menor quando usando nosso algoritmo. Terceiro, nosso algoritmo eficientemente atende as requisições das aplicações, respeitando seus níveis de prioridade, enquanto balanceia a carga de trabalho (workload) entre os nós da camada de borda.

1.6 Estrutura da tese

O Capítulo 2 apresenta os principais conceitos e a revisão da literatura relacionados aos algoritmos de sincronização de tempo e de alocação de recursos propostos. O Capítulo 3 apresenta nossa proposta de algoritmo de sincronização de tempo para a camada das coisas. O Capítulo 4 apresenta a avaliação experimental do algoritmo de sincronização proposto. O Capítulo 5 apresenta nossa proposta de alocação de recursos para a nuvem das coisas. O Capítulo 6 apresenta a avaliação experimental da proposta de alocação de recursos. O capítulo 7 apresenta as considerações finais, bem como os trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo apresenta os trabalhos mais relevantes das áreas referentes aos desafios de pesquisa desta tese. Inicialmente, nós apresentamos uma visão geral do problema de sincronização de tempo e os principais protocolos de sincronização de tempo em RSSFs. Em seguida, nós descrevemos uma visão geral sobre o problema de alocação de recursos na CoT e depois detalhamos os trabalhos mais relevantes que propõem soluções relacionadas a nossa proposta de gerenciamento de recursos na camada da borda.

2.1 Sincronização de Tempo na Camada das Coisas

Nas próximas seções, os principais conceitos sobre a sincronização de relógios na camada das coisas, e o estado da arte dos trabalhos relacionados sobre o tema são apresentados.

2.1.1 Principais Conceitos

Um relógio é um mecanismo que realiza a contagem de tempo por meio de um oscilador de cristal instalado no nó sensor. O oscilador é um circuito eletrônico que vibra continuamente e emite um sinal elétrico em uma dada frequência nominal f_i^0 para um nó i. Cada nó possui um registrador R_i, o qual é incrementado na frequência f_i^0. Desse modo, o valor do relógio de hardware presente em um nó i, no tempo físico t, pode ser definido como

$$H_i(t) = \left\lfloor \frac{R_i(t) - R_i(t_0)}{f_i^0} \right\rfloor + H_i(t_0) \tag{2.1}$$

em que t_0 pode ser considerado o momento que o nó é ligado e inicia o incremento

de R_i.

Ao realizar uma leitura em um relógio existem atrasos devido ao tempo gasto no caminho crítico da aplicação até o oscilador do sistema computacional. Logo, uma leitura realizada em dois nós i e j, simultaneamente, apresentarão valores diferentes. A diferença entre a leitura de tempo de dois relógios em um mesmo instante de tempo é chamada de offset relativo e é dada por

$$\theta_{ij}(t) = H_i(t) - H_j(t) \tag{2.2}$$

Dois relógios i e j também divergem devido à diferença entre a frequência real de cada um. O drift é a diferença entre a frequência real de dois relógios. Para os nós i e j, o drift relativo é definido como

$$\alpha_{ij}(t) = \frac{\theta_{ij}(t + \tau(t)) - \theta_{ij}(t)}{\tau(t)} \tag{2.3}$$

em que $\tau(t)$ é o intervalo entre duas leituras de relógio dos nós i e j no tempo t.

As variações do offset e do drift levam à perda de acurácia e precisão dos relógios. A acurácia é a capacidade que um mecanismo de sincronização possui de impedir que a diferença e o desvio entre dois ou mais relógios aumente. A precisão é a variabilidade com que o protocolo atinge determinada acurácia.

Para manter a acurácia dos relógios, os nós sensores trocam mensagens contendo informações de tempo entre si, de maneira que um nó seja capaz de ajustar seu relógio para algum tempo de referência. Este processo é chamado de sincronização de tempo, ou sincronização de relógios. A maioria dos protocolos de sincronização de relógios podem ser classificados como sender-receiver (SR) ou receiver-receiver (RR) como apontado por QIU *et al.* (2015).

Em protocolos SR apenas dois nós podem ser sincronizados de cada vez. Nesta metodologia, um nó receptor ajusta seu relógio baseado em um beacon de referência enviado por um nó emissor. Na metodologia SR, os dois nós podem sincronizar seus relógios por meio de um processo de troca de mensagens denominado two-way. Neste processo, o nó emissor inicia a sincronização enviando uma mensagem com o timestamp de envio para o nó receptor. O timestamp é o registro de tempo da ocorrência de um evento observado. O nó receptor responde com uma mensagem contendo o timestamp de recebimento e o timestamp de envio da mensagem de resposta. Quando o nó emissor recebe a mensagem de resposta, ele obtém o timestamp da resposta localmente. Com os timestamps coletados localmente, mais os valores de timestamps do nó receptor, o nó emissor pode calcular a diferença de tempo do relógio do nó receptor para o seu relógio, e ajustá-lo.

Em um protocolo RR, diversos nós podem sincronizar seus relógios com uma única mensagem do nó emissor (nó pai) para seus vizinhos (nós filhos). Inicial-

mente, o nó emissor envia uma mensagem de broadcast que serve como referência para seus nós filhos. Cada nó filho, que está em um nível inferior, armazena o tempo de chegada da mensagem e a propaga para seus nós vizinhos de mesmo nível. Durante esta propagação, os nós de mesmo nível utilizam os tempos de chegada das mensagens do nó pai e dos vizinhos para corrigir o valor de seus relógios.

Existe uma outra classe de protocolos que combina as características da metodologia SR e RR com o objetivo de reduzir o consumo energético da rede (QIU *et al.* (2018)). Em protocolos que combinam SR e RR, dois nós líderes sincronizam seus relógios com a metodologia SR, enquanto todos nós que são vizinhos simultaneamente dos nós líderes sondam as mensagens trocadas. Os nós vizinhos utilizam a metodologia RR com os valores de timestamp e tempo de chegada das mensagens para ajustarem seus relógios. A principal vantagem deste tipo de metodologia é que os nós vizinhos aos líderes não necessitam enviar qualquer mensagem para sincronizarem seus relógios.

Quando um nó sensor realiza uma troca de mensagens com outro nó, a fim de sincronizar seu relógio, existem cinco atrasos decorrentes da comunicação que são descritos a seguir.

Tempo de envio: tempo gasto para construir a mensagem e entregá-la à interface de rede, geralmente afetado pelo sistema operacional.

Tempo de acesso no envio: tempo necessário para a mensagem ser transmitida até o meio físico, sendo afetado principalmente pela contenção do canal.

Tempo de propagação: tempo para a mensagem transitar no meio físico desde o emissor até o receptor, sendo dependente principalmente da distância entre os dois nós.

Tempo de acesso no recebimento: tempo necessário para a mensagem sair do meio físico e ser processada na camada de enlace.

Tempo de recebimento: tempo necessário para construir a mensagem que chega da interface de rede e entregá-la à aplicação.

Tais atrasos são não-determinísticos e causam erros na correção do offset e do drift dos relógios. O erro de sincronização global de uma rede é a principal métrica para avaliar os erros de correção do drift e offset, e consiste no cálculo da diferença máxima entre as diferenças dos relógios de todos os nós sensores. O erro de sincronização é fortemente afetado pela distância em hops percorrida por uma mensagem de sincronização, desde um nó sensor até o nó raiz. LENZEN *et al.* (2008) mostraram a validade dessa observação e, além disso, demonstraram que o erro de sincronização aumenta em função do diâmetro da rede. Esse comportamento é denominado como problema o erro cumulativo e é um dos principais fatores que degradam a acurácia de um algoritmo de sincronização de relógios. Uma forma de mitigar o problema do erro cumulativo é escolher o nó de referência que minimize a distância dele para

os demais nós da rede, porém é um desafio atingir tal objetivo sem aumentar a quantidade de mensagens trocadas.

A maioria dos protocolos de sincronização não abordam o problema de selecionar o nó raiz (GANERIWAL *et al.* (2003), ELSON *et al.* (2003) etc) ou selecionam o nó raiz com menor número de identificador (MARÓTI *et al.* (2004), LENZEN *et al.* (2015), CHALAPATHI *et al.* (2019), JIA *et al.* (2019)). Algumas abordagens focadas em redução de consumo energético também não abordam o problema de seleção de nó raiz como os trabalhos de NOH e SERPEDIN (2007), NOH *et al.* (2008) e QIU *et al.* (2015). O protocolo RSync de QIU *et al.* (2018) também não aborda o erro cumulativo. Dessa forma, nós propomos um algoritmo de sincronização de tempo que é energeticamente eficiente como as abordagens mencionadas acima, porém fornece maior acurácia, pois mitiga o problema do erro cumulativo.

2.1.2 Revisão da Literatura

A seguir, nós descrevemos os protocolos de sincronização de tempo mais importantes de cada classe e os apresentamos em ordem cronológica. Ao final da descrição dos algoritmos, nós apresentamos a Tabela 2.1 onde as principais características dos protocolos de sincronização são apresentadas. Nesta tabela também constam as características do algoritmo de sincronização de tempo proposto, para que seja possível realizar uma comparação. As características apresentadas são a metodologia de avaliação, a acurácia e o consumo de energia. A seguir, os trabalhos relacionados com nossa proposta de sincronização de tempo para RSSFs.

Os autores ELSON *et al.* (2003) propuseram o RBS (Reference Broadcast Synchronization), um protocolo de sincronização RR que explora a característica de broadcast da RSSF. A rede é dividida em níveis a partir de um nó raiz, e um nó do nível superior envia uma mensagem de broadcast - um beacon de referência - para seus vizinhos no nível inferior, os quais informam o tempo de chegada do beacon entre si e atualizam seus relógios. A vantagem do RBS é eliminar a incerteza do atraso do emissor, pois o atraso de transmissão da mensagem do nó raiz é igual para todos os vizinhos que recebem o beacon de referência. O problema do RBS é que o número de mensagens cresce rapidamente por causa das diversas trocas de mensagens entre nós no mesmo nível.

Os autores GANERIWAL *et al.* (2003) propuseram o protocolo TPSN (Timing-sync Protocol for Sensor Networks) para sincronizar nós sensores. Este protocolo utiliza o método de sincronização SR, e consiste de dois passos principais. O primeiro passo é a inicialização, onde a rede é virtualmente organizada como uma árvore enraizada em um nó cujo relógio fornece o tempo de referência. No segundo passo, chamado sincronização, o nó raiz inicia um processo de sincronização com trocas de

mensagens two-way. O principal problema do TPSN é o alto número de troca de mensagens entre todos os nós da rede.

Os autores MARÓTI *et al.* (2004) propuseram o algoritmo FTSP, o qual utiliza a metodologia de sincronização SR . Os nós sensores fazem uso de um temporizador para enviar uma mensagem de sincronização para seus vizinhos periodicamente, porém não existe nenhuma coordenação na comunicação. Durante um longo período de execução, o FTSP gera um alto consumo energético, pois todos os nós trocam mensagens em todas as rodadas de re-sincronização.

Similar ao protocolo descrito acima, de GANERIWAL *et al.* (2003), HE (2008) propôs o protocolo TSBST (Time Synchronization Based on Spanning Tree for Wireless Sensor Networks). Este protocolo cria uma árvore de sincronização similar ao TPSN, no entanto, cada nó pai sincroniza com apenas um nó filho. Esta abordagem exige uma menor troca de mensagens quando comparada com o TPSN, porém o número de mensagens ainda é alto em redes densas e a acurácia é degradada de acordo com a escolha do nó filho em cada nível.

O algoritmo PBS (Pairwise Broadcast clock Synchronization) proposto por NOH e SERPEDIN (2007) tem o objetivo de reduzir o número de mensagens de sincronização a fim de ser eficiente em consumo de energia. O PBS combina os mecanismos de sincronização SR e RR dos algoritmos RBS de ELSON *et al.* (2003) e TPSN de GANERIWAL *et al.* (2003), respectivamente. Neste tipo de estratégia, dois nós líderes são escolhidos para realizar sincronização SR, realizando trocas de mensagens two-ray. Todos os nós periféricos que são vizinhos dos dois nós líderes sincronizam seus relógios por meio de sincronização RR. Nós periféricos sincronizam seus relógios apenas sondando a troca de mensagens entre os nós líderes e não necessitam enviar nenhuma mensagem de sincronização. Essa característica permite que o consumo energético seja reduzido, principalmente em redes densas, pois neste tipo de cenário existe uma grande quantidade de nós periféricos. A principal desvantagem do PBS é que todos os nós periféricos necessitam estar na intersecção da área de cobertura dos dois nós líderes para sincronizarem seus relógios.

O algoritmo GPA (Groupwise Pair selection Algorithm) proposto por NOH *et al.* (2008) estende o algoritmo PBS, pois permite sincronizar mais do que um grupo de nós controlados por dois nós líderes. Os resultados do trabalho mostram que o GPA é eficiente em reduzir o consumo energético durante a operação regular do algoritmo, porém é custoso selecionar o conjunto de pares de nós líderes, principalmente em cenários de falha.

Os autores SKOG e HANDEL (2010) propuseram um estimador aproximado de máxima vizinhança para aumentar a acurácia de um esquema de sincronização baseado na metodologia SR. Embora a estratégia seja capaz de reduzir o erro de sincronização, devido à grande troca de mensagens ela não leva em conta os limitados

recursos da rede de sensores.

O algoritmo OPRBS desenvolvido pelos autores JAIN e SHARMA (2011) foi proposto para reduzir o número de mensagens do protocolo RBS. Para alcançar este objetivo o OPRBS aumenta a área de cobertura do nó sensor, por meio do aumento da potência do sinal de rádio, para que uma quantidade maior de nós receba o beacon de sincronização simultaneamente. O problema do OPRBS é que o aumento da potência do sinal exige um maior consumo de energia da antena do nó sensor.

Os autores DJENOURI *et al.* (2013) propuseram o algoritmo R4Syn baseado no RBS. O algoritmo proposto pelos autores distribui o papel de nó de referência, o qual existe no protocolo RBS, entre todos os nós da rede. Para realizar essa distribuição do papel de nó de referência, o beacon de sincronização contém os últimos timestamps dos nós vizinhos do nó de referência. Esta abordagem reduz a quantidade de mensagens de sincronização quando comparada com o RBS, porém, devido as mensagens de sincronização conterem todos os timestamps dos últimos vizinhos do nó de referência, então as mensagens de sincronização possuem um tamanho maior, o que gera mais dados trafegando na rede.

BENZAÏD *et al.* (2014) propuseram o algoritmo Spirt, que usa as metodologias SR e RR, similar ao PBS para sincronizar os nós sensores. A melhoria do Spirt consiste em sincronizar nós periféricos que não estão na intersecção das áreas de transmissão dos dois nós líderes. O Spirt realiza esta sincronização propagando os timestamps dos nós líderes por mais do que um hop de distância. Os autores alegam que o número de mensagens de sincronização é reduzido, no entanto, o aumento do tamanho da mensagem gera mais dados trafegados na rede quando comparado com o PBS.

No trabalho de LI *et al.* (2014), os autores propuseram uma estratégia para reduzir logicamente o diâmetro da rede para reduzir o impacto erro cumulativo. No entanto, ao invés de selecionar o nó raiz, os autores formulam um problema de otimização para criar um overlay virtual da rede, o qual define os caminhos ótimos de cada nó para o nó raiz. Os autores executam o FTSP sobre o overlay, reduzindo o erro de sincronização do algoritmo. A abordagem proposta por LI *et al.* (2014) é custosa em termos computacionais e a metodologia de sincronização também exigi um grande número de mensagens de sincronização.

LENZEN *et al.* (2015) propuseram o protocolo de sincronização PulseSync que é similar ao FTSP. No PulseSync, ao invés de cada nó configurar um temporizador para enviar mensagens de sincronização periodicamente, o nó encaminha imediatamente a mensagem de sincronização recebida por seu nó pai. Este encaminhamento rápido reduz o atraso de envio da mensagem, aumentando a acurácia dos relógios. No entanto, pode não ser adequado para sistemas de low duty, pois neste tipo de sistema um nó necessita aguardar estar em um estado ativo para iniciar a transmissão

de alguma mensagem. Além disso, o PulseSync pode apresentar um alto consumo energético, pois todos os nós transmitem mensagens de sincronização em todas as rodadas de re-sincronização.

Os autores QIU *et al.* (2015) propuseram o algoritmo STETS que sincroniza os nós incorporando as estratégias SR e RR similar ao GPA. No entanto, no STETS a escolha dos nós líderes (chamados nós backbones) é realizada por meio de um mecanismo de expiração de temporizador, o qual a torna mais eficiente. O problema do algoritmo STETS é que ele não é robusto o suficiente para sincronizar todos os nós da rede, gerando alguns nós isolados.

O algoritmo TATS (Time-of-flight Aware Time Sinchronization) foi proposto por LIM *et al.* (2016) para melhorar a acurácia do PulseSync, pois o TATS compensa o atraso de propagação no mecanismo de correção de relógio. O TATS apresenta uma alta acurácia, porém devido a todos os nós trocarem mensagens de sincronização durante todas as re-sincronizações, ele pode apresentar um alto consumo energético durante um longo período de operação.

Os autores BENZAÏD *et al.* (2017) propuseram o algoritmo E-Spirt, o qual melhora o Spirt, pois permite sincronização multihop. Ele é similar ao TPSN, funcionando em uma etapa de criação de árvore de sincronização e depois sincronização dos nós sensores. O E-Spirt necessita de mais tráfego de dados como o Spirt, e a sincronização pode ser degradada, pois um nó sensor utilizada o timestamp de rodadas anteriores para sincronizar seu relógio. Além disso, é custoso executar o algoritmo que seleciona todos os pares de nós líderes

O algoritmo RSync foi proposto por QIU *et al.* (2018) para atenuar o problema do algoritmo STETS, de maneira que para redes densas todos os nós são capazes de sincronizar seus relógios. O RSync contorna o problema de nós isolados do STETS adicionando um novo temporizador que aciona um mecanismo de anexação do nó isolado à rede sincronizada. O RSync ainda seleciona, a cada rodada de re-sincronização, o vizinho do nó raiz que possui a maior quantidade de energia restante como novo nó raiz. Tal estratégia de seleção de nó raiz não aborda o problema do erro cumulativo.

O algoritmo C-TPSN (Timing-Sync Protocol for Sensor Networks of Chain), dos autores JIA *et al.* (2019), foi proposto para melhorar o desempenho do TPSN. A proposta dos autores consiste em melhorar a fase do TPSN em que é gerada a árvore da rede. Eles fazem isso criando clusters de nós, a partir do nó raiz, que minimizam a profundidade da árvore gerada. A proposta reduz o número de níveis, porém ela ainda sofre do problema de todos os nós necessitarem enviar mensagens durante todas as re-sincronizações.

Os autores CHALAPATHI *et al.* (2019) propuseram o protocolo E-SATS (Efficient and Simple Algorithm for Time Synchronization) que utiliza metodologia SR

para sincronizar uma rede clusterizada. O E-SATS possui uma fase inicial em que os nós são organizados em clusters e uma fase de sincronização em que os membros de cluster sincronizam seus relógios com o relógio do nó head do cluster. A sincronização consiste em uma quantidade pré-determinada de troca de mensagens two-ray entre o head do cluster (via broadcast) e os membros de cluster (via mensagens unicast). O E-SATS reduz a quantidade de mensagens quando comparado com o SATS (Simple Algorithm for Time Synchronization) dos autores CHALAPATHI *et al.* (2016), o qual consiste em um protocolo similar, porém possui a limitação do head do cluster enviar mensagens unicast para cada vizinho, não aproveitando a natureza broadcast do canal de comunicação wireless. O E-SATS sofre de problemas similares ao TPSN, pois exige que todos os nós enviem mensagens durante cada rodada de sincronização, o que aumenta o consumo energético.

Em resumo, os algoritmos propostos por MARÓTI *et al.* (2004), LENZEN *et al.* (2015) e LIM *et al.* (2016) são algoritmos eficientes em acurácia, porém eles conseguem diminuir o erro de sincronização em sacrifício do consumo energético, pois todos os nós devem manter o envio de mensagens periódicas durante todo o tempo de operação da rede. Os algoritmos desenvolvidos pelos autores MARÓTI *et al.* (2004), LENZEN *et al.* (2015) e LIM *et al.* (2016) também utilizam uma técnica trivial de selecionar o nó raiz como aquele que possui o menor número de identificador. Tal abordagem não possui impacto no número de mensagens de sincronização utilizadas pelo protocolo ou na acurácia da sincronização da rede.

Os protocolos de sincronização como aqueles desenvolvidos por GANERIWAL *et al.* (2003), HE (2008), SKOG e HANDEL (2010), CHALAPATHI *et al.* (2016) JIA *et al.* (2019) e CHALAPATHI *et al.* (2019) também necessitam que todos os nós realizem trocas de mensagens two-way durante todo o tempo de operação da rede, e não visam tratar o problema do erro cumulativo, pois selecionam o nó raiz aleatoriamente. As abordagens que fazem uso de sincronização RR como as projetadas pelos autores ELSON *et al.* (2003), JAIN e SHARMA (2011) e DJENOURI *et al.* (2013) também sofrem com uma quantidade grande de mensagens sendo trocadas entre nós de mesmo nível na rede. Tanto as abordagens que sincronizam os relógios pela metodologia SR, quanto as abordagens que utilizam a metodologia RR escolhem o nó raiz aleatoriamente. Ainda, a abordagem de LI *et al.* (2014), apesar de tratar o problema do erro cumulativo, necessita otimizar toda topologia da rede, o que gera um alto custo computacional, além de realizar sincronização SR, que não é eficiente energeticamente.

Os algoritmos desenvolvidos pelos autores NOH e SERPEDIN (2007), NOH *et al.* (2008), QIU *et al.* (2015), BENZAÏD *et al.* (2014) e BENZAÏD *et al.* (2017), que incorporam os mecanismos de sincronização SR e RR, são eficientes energeticamente, porém escolhem o nó raiz aleatoriamente. O RSync de QIU *et al.* (2018) é o protocolo

que representa o estado da arte dos protocolos de sincronização de tempo que são energeticamente eficientes, pois além de não escolher o nó raiz aleatoriamente, ainda corrige o problema de nós isolados dos outros protocolos.

O RSync exige uma menor quantidade de troca de mensagens e não utiliza uma abordagem aleatória para selecionar o nó raiz. Ele seleciona como nó raiz o nó vizinho que possui a maior quantidade de energia restante. O problema com a estratégia de seleção de nó raiz do RSync é que o consumo energético não é reduzido, pois todos os nós enviam a mesma quantidade de mensagens, inclusive o raiz. O RSync também não aborda o problema do erro cumulativo.

O protocolo proposto nesta tese seleciona o nó raiz ótimo de maneira que a distância em hops de cada nó para o nó raiz seja minimizada. Ao escolher o nó raiz que possui essa característica, é possível aumentar o intervalo de re-sincronização e ainda manter a acurácia do protocolo. O benefício de aumentar o intervalo de ressincronização é que, no longo prazo, o número de mensagens trocadas entre os nós é reduzido, aumentando, assim, o tempo de vida da rede de nós sensores. O protocolo de sincronização de tempo proposto nesta tese seleciona o nó raiz ótimo sem necessidade de aumentar a quantidade de mensagens trocadas na rede, o que evita um impacto no consumo energético.

Trabalho	**SR**	**RR**	**SR e RR**	**Acur.**	**Ene.**	**Err. Cum.**
GANERIWAL *et al.* (2003)	✓			✓		
ELSON *et al.* (2003)		✓		✓		
MARÓTI *et al.* (2004)	✓			✓		
NOH e SERPEDIN (2007)			✓		✓	
LENZEN *et al.* (2008)	✓			✓		
NOH *et al.* (2008)			✓		✓	
JAIN e SHARMA (2011)		✓		✓		
DJENOURI *et al.* (2013)		✓		✓		
BENZAÏD *et al.* (2014)			✓		✓	
LI *et al.* (2014)	✓					✓
LENZEN *et al.* (2015)	✓			✓		
QIU *et al.* (2015)			✓		✓	
CHALAPATHI *et al.* (2016)	✓			✓		
LIM *et al.* (2016)	✓			✓		
BENZAÏD *et al.* (2017)			✓		✓	
QIU *et al.* (2018)			✓		✓	
CHALAPATHI *et al.* (2019)	✓			✓		
JIA *et al.* (2019)	✓			✓		
Algoritmo proposto			✓	✓	✓	✓

Tabela 2.1: Resumo das principais características dos trabalhos sobre gerenciamento de recursos

2.2 Gerenciamento de Recursos na Nuvem das Coisas

Os conceitos principais e o estado da arte dos trabalhos relacionados ao gerenciamento de recursos na nuvem das coisas são apresentados nas próximas seções.

2.2.1 Principais Conceitos

A CoT é composta de dispositivos da IoT, os quais são objetos físicos equipados com recursos de computação e comunicação em rede e que fornecem provisão de serviços de sensoriamento, computação e autuação sob demanda. Tais dispositivos são pobres em recursos, de maneira que aplicações que exigem um alto poder computacional são encaminhadas para serem processadas na nuvem. A nuvem é o local centralizado, composta por datacenters com grande capacidade computacional e de comunicação, e que fornece serviços em um ambiente altamente virtualizado. Dessa forma, a arquitetura da CoT pode ser composta de duas camadas: camada das coisas e camada da nuvem.

Os dispositivos da IoT, bem como os dispositivos que executam as aplicações, são fisicamente distantes da nuvem, o que acarreta em grande atraso de comunicação e ineficiente uso da infraestrutura da CoT. A fim de reduzir a latência de comunicação das aplicações da CoT, dispositivos da camada das coisas como dispositivos de identificação de rádio frequência (RFID - Radio Frequency Identification), dispositivos controlados por humanos (smartphones ou tablets, por exemplo), dispositivos de rede (roteadores e switchs inteligentes, por exemplo), máquinas ricas em recursos (cloudlets) e computadores de tamanho de cartão de crédito, têm sido usados para trazer computação para o que é chamado, a borda da rede.

Esse tipo de dispositivo é chamado de nó de borda e possui função similar a da nuvem de fornecer serviço para as aplicações da CoT. Os dispositivos presentes na borda da rede são conectados com a nuvem e podem estar conectados entre si em uma rede descentralizada. Os nós de borda estão mais próximos geograficamente das aplicações e dos dispositivos da IoT e formam uma rede descentralizada, mais parecida com a arquitetura da camada das coisas. Por esses motivos, os nós de borda podem fornecer serviços respeitando as restrições de latência, e demais requisitos das aplicações da CoT, de maneira mais efetiva. A adoção da computação na borda da rede faz emergir o novo paradigma de computação de borda (edge computing). Neste paradigma, a CoT é composta de três camadas: camada das coisas, camada de borda (borda da rede) e camada da nuvem.

Assim como na nuvem, os nós da camada de borda são fortemente baseados em virtualização, o que proporciona o desacoplamento entre a infraestrutura física dos

dispositivos da IoT e as aplicações, via a representação de nós virtuais. Cada nó virtual pode executar uma ou mais requisições de aplicação, as quais são conjuntos de instruções com as demandas dos recursos da CoT.

As requisições das aplicações da CoT possuem requisitos que características funcionais do sistema da CoT e requisitos de Qualidade de Serviço (QoS). A latência e o tempo de resposta da aplicação são alguns dos principais requisitos para a maioria dos domínios de aplicações da CoT. Outro requisito que se destaca, principalmente em sistemas de sensoriamento, é o frescor do dado, pois a integração de dados com diferentes valores de frescor é um problema que dificulta a execução da aplicação. Alguns importantes requisitos da CoT são relacionados as diferentes prioridades da aplicação, consumo energético, diversidade de conectividade, escalabilidade e segurança, não se restringindo a esses.

Os nós que compõem todo o ecossistema da CoT podem estar organizados de diferentes maneiras. Em uma arquitetura de duas camadas, os dispositivos da IoT acessam a nuvem por meio da Internet. Na arquitetura de três camadas, existem mais complexas formas de organização dos participantes da CoT. Nesta arquitetura, os dispositivos da IoT se conectam por meio de um ponto de acesso com os nós de borda. Geralmente o ponto de acesso está na camada de borda.

Nesta tese, nós consideramos três tipos de organização dos nós na camada de borda, os quais chamamos de Vertical, Horizontal e Hierárquico. A organização Vertical corresponde a uma arquitetura em que nós de borda não possuem conexões entre si e são conectados apenas com a nuvem. A organização Horizontal denota uma arquitetura em que os nós de borda possuem conexão entre si em uma topologia em linha, além de conexão com a nuvem. A organização Hierárquico corresponde a uma arquitetura em que os nós de borda possuem conexões entre si, as quais são dadas por qualquer topologia de um grafo conectado, e apenas alguns nós selecionados possuem conexão com a nuvem.

Na camada da nuvem, os datacenters que a compõem representam um conjunto de recursos centralizados e são conectados por meio da Internet com a camada de borda.

Na próxima na Seção 2.2.2, nós consideramos os conceitos discutidos acima para apresentar os principais trabalhos que possuem relação com a abordagem de gerenciamento de recursos proposta nesta tese.

2.2.2 Revisão da Literatura

Nesta seção, nós descrevemos e comparamos os trabalhos sobre alocação de recursos que estão diretamente relacionados com nossa proposta, considerando os seguintes aspectos: (i) nível de centralização, (ii) organização da camada de borda,

(iii) requisitos das aplicações atendidos, e (iv) a heterogeneidade das aplicações. Ao fim da seção, a Tabela 2.2 apresenta uma classificação resumida das principais características de cada trabalho. O nível de centralização denota se o processo de decisão sobre a alocação de recursos ocorre de maneira centralizada, parcialmente descentralizada ou descentralizada. Os requisitos atendidos pelas aplicações descritos na tabela são: latência, frescor do dado e prioridade da requisição. Alguns trabalhos podem atender outros requisitos, como consumo de energia ou uso de recursos, no entanto nós colocamos na tabela apenas os requisitos relacionados ao nosso trabalho. A organização da camada de borda pode assumir os valores Vertical, Horizontal ou Hierárquico. Por fim, a heterogeneidade das aplicações indica se a proposta considera diferentes tipos de requisições de aplicações ou requisições homogêneas.

DO *et al.* (2015) assumem que os nós de computação fog (nó de borda) são colocados na borda da rede para entregar serviços de streaming de vídeo. Os nós fog agregam demandas de vídeo de usuários próximos e o objetivo da estratégia desenvolvida pelos autores é reduzir a latência de serviço e aumentar a utilidade de serviço. Os autores propuseram um algoritmo proximal distribuído para o problema conjunto da alocação de recursos e da minimização da métrica pegada de carbono (footprint carbon) para o serviço de streaming de vídeo na camada de borda. Nosso algoritmo de alocação de recursos difere da abordagem dos autores porque ele segue um modelo baseado em agentes econômicos Ainda, no trabalho dos autores, a alocação de recursos é parcialmente descentralizada, pois um datacenter na nuvem possui responsabilidades de coordenação da entrega das requisições de vídeos, enquanto nossa abordagem é descentralizada. Finalmente, os autores consideram apenas a latência como requisito da aplicação, enquanto nosso algoritmo aborda outros requisitos além da latência, como frescor do dado e prioridade das requisições. O trabalho os autores também não aborda a heterogeneidade das aplicações.

DENG *et al.* (2016) formularam um problema de alocação de recursos em sistemas borda-nuvem, o qual tem como requisitos principais a redução da latência das aplicações e o consumo de energia. O trabalho se propõe a resolver os seguintes problemas: o trade-off entre o consumo de energia e o atraso na camada de borda, e o trade-off entre o consumo de energia na nuvem e a minimização do atraso de comunicação para o envio de requisições dos nós fog para a nuvem. O trabalho de DENG *et al.* (2016) trata da latência e do consumo de energia na camada de borda, no entanto, diferente de nosso trabalho, a proposta dos autores não trata outros requisitos como o frescor do dado e as prioridades das aplicações. Além disso, embora a abordagem dos autores seja descentralizada, os nós da camada de borda não realizam colaboração de requisições. Por outro lado, nosso algoritmo de alocação de recursos também é descentralizado e permite a colaboração entre nós de borda. Por

fim, o trabalho dos autores não considera que as aplicações possuam níveis de prioridades diferentes, tratando todas as requisições enviadas para a camada de borda de maneira equivalente.

DEHURY e SAHOO (2016) projetaram um framework para gerenciar requisições da IoT de tempo real. A proposta dos autores é baseada em virtualização e consiste de dois componentes principais: o Fornecedor de Serviço e o Middleware. O Fornecedor de Serviço é responsável por gerenciar os recursos físicos da nuvem, enquanto o Middleware trabalha sob demanda e entrega os recursos disponíveis para as aplicações que os requisitam. O principal requisito tratado pelos autores é a latência das aplicações e, além disso, a proposta provê serviço diferenciado de acordo com as prioridades das aplicações. A proposta de DEHURY e SAHOO (2016) é centralizada na nuvem e implementada em uma arquitetura de duas camadas, a qual é diferente de nossa proposta que é descentralizada e considera a camada de borda da rede. Os autores tratam as prioridades das aplicações, porém de maneira diferente de nossa abordagem, a qual considera o preço pago pelo usuário como valor de prioridade.

O trabalho de ZHANG *et al.* (2017) formula o processo de alocação de recursos em uma arquitetura de três camadas como o problema de compra de recursos por dispositivos da IoT, os quais eles chamam de DSSs (Data Service Subscribers), e o problema de precificação de recursos de nós fog e datacenters na nuvem, os quais são denominados DSOs (Data Service Operators). Os autores, então, abordam a alocação de recursos na borda da rede como o problema de emparelhamento de DSOs e nós fog e o problema de emparelhamento de DSSs e nós fog. Um ponto de convergência da proposta dos autores com nosso trabalho é que o algoritmo de ZHANG *et al.* (2017) também precifica os recursos dos nós e borda, no entanto eles utilizam uma abordagem baseada em teoria dos jogos para resolver o problema de alocação, enquanto nós seguimos um modelo baseado em agentes econômicos. O trabalho dos autores tenta otimizar o uso computacional dos nós fog, porém não considera outros requisitos como a latência. Finalmente, o trabalho assume que todos os nós fog são capazes de produzir cada tipo de dado, ou seja, os autores consideram que as aplicações são homogêneas em termos dos tipos de dados.

LI *et al.* (2017) propuseram uma arquitetura de três camadas e avaliaram esquemas de alocação de tarefas, variando a porcentagem de requisições alocadas em cada camada. Na abordagem proposta pelos autores, a nuvem é responsável por tarefas que exigem computação intensiva de dados e grande quantidade de espaço de armazenamento. Na camada da borda, as requisições atendidas são aquelas que requerem processamento de tempo real. As requisições dos usuários passam todas pela nuvem, mesmo aquelas que são atendidas na camada e borda. O algoritmo de alocação de recursos é parcialmente descentralizado, devido ao fato das requisições serem provenientes da nuvem, a qual realiza uma coordenação inicial. Na abordagem

dos autores não há colaboração entre os nós da camada de borda e as requisições das aplicações são homogêneas, de maneira que sempre possuem o mesmo tipo de dado. O algoritmo dos autores aborda o problema da latência indiretamente, já que o requisito atendido é o balanceamento de requisições de aplicações de tempo-real críticas na camada de borda.

O trabalho de ALI *et al.* (2018) considera o problema de selecionar uma cloudlet, considerada como um nó de borda pela proposta, para se associar a um dispositivo da IoT, usando uma conexão sem fio. O objetivo do algoritmo é escolher uma cloudlet para cada dispositivo da IoT, maximizando o balanceamento da carga de trabalho (workload) e minimizando a latência das aplicações. Para isso, os autores propuseram um algoritmo distribuído para os dispositivos IoT encontrarem soluções para o problema de seleção das cloudlets. No trabalho dos autores, embora haja comunicação entre as cloudlets, não existe colaboração entre os nós e não há diferenciação quanto ao tipo de dado das requisições. Nosso trabalho se difere, pois nosso algoritmo de alocação de recursos atende outros requisitos como o frescor do dado e as prioridades das aplicações, e considera aplicações heterogêneas. Além disso, nosso alocador de recursos aloca nós virtuais para atender as requisições, o que proporciona o desacoplamento entre as aplicações e a infraestrutura dos dispositivos que compõem a camada de borda, enquanto a abordagem de selecionar uma cloudlet não proporciona tal desacoplamento.

BITAM *et al.* (2018) estudaram o problema de melhor alocação de tarefa para nós da camada de borda, em que o tempo de execução e a quantidade de memória requirida são os parâmetros a serem minimizados pelo modelo dos autores. O problema formulado pelos autores é resolvido por uma abordagem de otimização bio-inspirada baseada em inteligência de enxame, um algoritmo de otimização de abelhas. Diferente da proposta baseada na heurística evolutiva de enxame de abelhas, nosso alocador de recursos parte de uma abordagem baseada em mercado, atribuindo a cada nó de borda a função de agente econômico e a cada alocação de requisição para nó virtual como o bem negociado. A principal desvantagem do trabalho dos autores é a natureza centralizada da solução, pois o algoritmo evolutivo é executado em um nó na nuvem, o qual decide em qual nó fog cada requisição será atendida. Além disso, o problema formulado pelos autores minimiza o uso dos recursos de processamento e memória, porém não a latência diretamente.

Os autores LI *et al.* (2018) propuseram um esquema para alocar máquinas virtuais para atendimento de aplicações da CoT, com o objetivo de maximizar a utilização dos links de rede e evitar congestão dos canais de comunicação. Os autores modelam as requisições das aplicações como uma topologia de recursos, formulando um problema de otimização. Este problema é resolvido por um algoritmo heurístico baseado em teoria dos grafos proposto pelos autores. O trabalho dos autores foca

principalmente em evitar gargalos na rede, e não atende a algum requisito de QoS especificamente. Diferente de nossa abordagem, o trabalho dos autores é centralizado e considera uma arquitetura de duas camadas, em que os dispositivos da IoT se comunicam com um datacenter na nuvem. Além disso, o algoritmo dos autores se destina a otimizar principalmente parâmetros de rede, enquanto nossa abordagem trata alguns requisitos de aplicações da CoT.

YOUSEFPOUR *et al.* (2018) propuseram um algoritmo de alocação de recursos para minimizar o atraso, o qual aborda as interações de dispositivos da IoT para nuvem e de nós fog para a nuvem, além de empregar o compartilhamento de carga entre nós fog. A abordagem dos autores possui um modo de operação centralizado em que a orquestração dos recursos ocorre na nuvem e um modo de operação descentralizado, em que os nós fog distribuem sua informação de estado com seus vizinhos a fim de escolher o melhor vizinho para atender uma requisição. O algoritmo proposto pelos autores considera dois tipos de requisições, leves e pesadas, e a partir desta definição, decide onde a requisição é atendida. O trabalho de YOUSEFPOUR *et al.* (2018) pode atuar de maneira descentralizada, o qual possui semelhança com nosso algoritmo de alocação de recursos, pois cada nó da camada de borda toma a decisão de alocar uma requisição acessando a informação de seus vizinhos. O trabalho dos autores trata da latência das aplicações, porém não considera outros requisitos como o frescor do dado e a prioridade das requisições. Outro aspecto do trabalho dos autores é que as requisições também são homogêneas, sendo possível uma requisição ser atendida por qualquer nó de borda.

Os autores WANG *et al.* (2018) propuseram um mecanismo de gerenciamento de recursos para tratar o problema de multi-tenancy para computação na borda. A abordagem proposta usa provisionamento vertical juntamente com gerenciamento de prioridades que podem ser estáticas ou dinâmicas, com o objetivo de múltiplas cargas de trabalho co-existirem hospedadas no mesmo nó de borda para reduzir a violação de taxas de objetivos de nível de serviço e a latência. O trabalho de WANG *et al.* (2018) possui um ponto de convergência com nossa abordagem, no sentido que ele trata as prioridades a partir do valor pago pelo usuário em usar o sistema. No entanto ao invés e usar um modelo de agentes econômicos para modelar o problema de alocação, os autores fazem uso de uma abordagem baseada em jogos de matching.

No trabalho de VASCONCELOS *et al.* (2019), os autores modelam a camada de borda com dois tipos de nó, os nós fog e nós mists. Nós fog possuem uma conexão direta com a Internet e podem atuar como gateways para um dispositivo da IoT, enquanto os demais dispositivos são nós mist. Os parâmetros a serem otimizados pelo algoritmo são a latência, a largura de banda e o custo financeiro para alocar um conjunto de recursos para uma requisição. O trabalho de VASCONCELOS *et al.* (2019) modela o problema de alocação de recursos como um grafo onde os vértices

são os dispositivos e as arestas são os links de comunicação, e os autores propõem um algoritmo distribuído greedy baseado no algoritmo de Djkstra para definir um ranking dos melhores nós para alocar uma requisição. A proposta dos autores trata o problema da latência, largura de banda e custo financeiro da alocação. A abordagem de VASCONCELOS *et al.* (2019) se assemelha ao nosso algoritmo no sentido que fornece um modelo do custo financeiro da alocação de recurso. No entanto, nossa abordagem usa o custo financeiro para estabelecer prioridades entre as requisições. Nosso algoritmo também considera outros parâmetros como o frescor do dado e a heterogeneidade das aplicações. O algoritmo de VASCONCELOS *et al.* (2019) é descentralizado e trata requisições homogêneas, pois os nós fog ou mist sempre são capazes de atender as requisições. A abordagem dos autores também permite a colaboração entre entre nós fog ou mist.

Os autores ALRAWAHI *et al.* (2019) propuseram um modelo de mercado para representar recursos como commodities, e um alocador de recursos que habilita a troca das commodities, precificando-as por meio de um modelo ao estilo pague-pelo-uso. A abordagem dos autores é modelada em uma arquitetura de duas camadas, e o principal objetivo do algoritmo de alocação consiste em distribuir os recursos otimamente, ao mesmo tempo que atende os requisitos de custo financeiro do recurso, tempo de resposta, consumo de energia, tolerância à falha e cobertura de recurso. O modelo proposto pelos autores consiste de um problema multiobjetivo onde um grupo de consumidores (aplicações) são atendidos por um grupo de fornecedores de recursos, de maneira que os consumidores possuem requisitos de QoS a serem respeitados. Para resolver o problema os autores avaliam o uso de três algoritmos evolucionários já existentes. De maneira similar à proposta dos autores, nosso algoritmo também faz uso do conceito de mercado econômico para resolver o problema de alocação, porém nossa abordagem parte da premissa que os recursos são virtualizados e que os bens econômicos trocados são nós virtuais alocando requisições. Isso permite o desacoplamento da infraestrutura física das aplicações. Além disso, em nosso modelo, os nós de borda atuam como agentes econômicos, o que é diferente da proposta de comoditização proposta pelos autores. Ainda, nossa abordagem tem como premissa uma arquitetura da CoT de três camadas. Finalmente, os requisitos de QoS atendidos em comum entre nossa proposta e a proposta dos autores são o custo financeiro e tempo de resposta (latência da aplicação).

Os autores SANTOS *et al.* (2019) propuseram um algoritmo heurístico para resolver um problema de otimização para alocação de recursos na camada de borda. O algoritmo analisa as requisições de aplicações, identifica as requisições que são comuns para múltiplas aplicações, executa a requisição apenas uma vez e compartilha o dado, por meio do processo de virtualização, entre as diversas aplicações. O algoritmo possui uma fase de decisão centralizada executada na nuvem, onde um

nó virtual de um dispositivo na borda é alocado para atender a requisição, e uma fase descentralizada, executada pelo nó de borda, onde é determinado se o dado requisitado deve ser atualizado na camada das coisas. O problema da latência é tratado pelos autores, bem como o problema do frescor do dado para aplicações heterogêneas. A redução do consumo de energia também é um requisito explorado, devido ao compartilhamento de dados dos nós virtuais. Nosso algoritmo se difere da abordagem dos autores, pois além de tratar do problema do frescor do dado de aplicações heterogêneas, ele também respeita as prioridades das requisições das aplicações. Além disso, a proposta dos autores é parcialmente centralizada, ao passo que nosso algoritmo é completamente descentralizado.

	Arquitetura		**Nível de Centralização**			**Organização da C. de Borda**			**Requisitos**			**Heterogeneidade**	
Trabalho	**2 camadas**	**3 camadas**	**Centralizado**	**Descentralizado**	**Parc. Desc.**	**Vertical**	**Horizontal**	**Hierárquico**	**Latência**	**Frescor do Dado**	**Prioridade**	**Heterogêneo**	**Homogêneo**
DO *et al.* (2015)		✓			✓	✓			✓			✓	
DENG *et al.* (2016)		✓		✓		✓			✓				✓
DEHURY e SAHOO (2016)	✓		✓			✓			✓		✓	✓	
ZHANG *et al.* (2017)		✓			✓	✓							✓
LI *et al.* (2017)		✓			✓	✓			✓				✓
ALI *et al.* (2018)		✓		✓			✓		✓				✓
BITAM *et al.* (2018)		✓	✓			✓			✓				✓
LI *et al.* (2018)	✓		✓			✓							✓
YOUSEFPOUR *et al.* (2018)		✓		✓				✓	✓				✓
WANG *et al.* (2018)		✓	✓			✓			✓		✓		✓
SANTOS *et al.* (2019)		✓			✓	✓			✓	✓		✓	
ALRAWAHI *et al.* (2019)	✓		✓			✓			✓			✓	
VASCONCELOS *et al.* (2019)		✓	✓					✓	✓				✓
Algoritmo proposto		✓		✓				✓	✓	✓	✓	✓	

Tabela 2.2: Resumo das principais características dos trabalhos do estado da arte sobre gerenciamento de recursos.

Capítulo 3

Sincronização de Tempo na Camada das Coisas

Neste capítulo, nós apresentamos a proposta de algoritmo para sincronização de tempo para sistemas da CoT, os quais usam a RSSF como infraestrutura de sensoriamento e comunicação da camada das coisas. Nosso algoritmo realiza a sincronização dos relógios dos nós sensores da RSSF utilizando as metodologias SR e RR em conjunto, com o objetivo de ser eficiente energeticamente. Além disso, ele seleciona o nó de referência da rede com o objetivo de mitigar o problema do erro cumulativo, aumentando assim a acurácia da sincronização. A seleção do nó raiz faz uso de temporizadores para estimar propriedades da rede e evitar a troca de informações sobre a topologia. Na Seção 3.1 nós apresentamos as definições preliminares para a modelagem do problema. Na Seção 3.2 nós descrevemos o modelo de relógio e a modelagem do problema de seleção do nó raiz. Na Seção 3.3, nós apresentamos nosso algoritmo de sincronização.

3.1 Definições Preliminares

A rede de sensores sem fio é modelada como um grafo não direcionado $G = (V, E)$, onde V é o conjunto de $n = |V|$ nós e E é o conjunto de arestas que representam links de comunicação sem fio entre nós sensores. O conjunto de nós dentro da área de transmissão (área de alcance do sinal de rádio) de um nó $i \in V$ é o conjunto de seus vizinhos e é denotado como $N_i = \{j \in V | (i,j) \wedge (j,i) \in E\}$. Um nó único da rede $r \in V$ é denotado como o nó raiz e ele possui o papel de inicializar a sincronização. O custo de uma aresta que conecta dois vértices adjacentes $i \wedge j \in V$ é a distância em hop (salto) entre i e j e é dada pela função $w(i,j)$. Quando $w(i,j) = 1$ significa que i e j estão na área de transmissão um do outro. Cada nó i é categorizado com um tipo $\tau_i \in \{UN, BN, PN\}$, o qual define três categorias de

nós, representando estados de sincronização chamados:

- Undefined Node (UN) – é o estado inicial de cada nó. Esse estado significa que o nó não foi sincronizado.
- Backbone Node (BN) – nós que realizam troca de mensagens e sincronizam seus relgios por meio da metodologia SR.
- Passive node (PN) – nós que não trocam mensagens e sincronizam seus relógios por meio da metodologia RR. Esses nós sincronizam seus relógios sondando as mensagens trocadas pelos nós BN.

Cada nó i possui três temporizadores: $seltimer_i$, $synctimer_i$ e $pulltimer_i$. O $seltimer_i$ é utilizado para indicar ao nó i o momento em que ele pode ser considerado o nó raiz da próxima rodada de ressincronização. O $synctimer_i$ tem a função de iniciar a troca de mensagens two-way entre um nó filho e um nó pai. O $pulltimer_i$ tem o objetivo de fazer com que um nó UN possa se sincronizar com algum vizinho. Os nós podem trocar cinco tipos de mensagens com os objetivos de selecionar o nó raiz e sincronizar os relógios dos nós sensores: $INIT\_D$, $INIT\_S$, $SYNC$, ACK e $PULL$. Para uma mensagem msg, existem 9 diferentes campos, de maneira que os campos 5 e 6 podem representar valores diferentes, a depender do tipo de mensagem. Durante a Fase de Seleção (descrita na Seção 3.3, todos os nove campos são utilizados), no entanto durante a Fase de Sincronização (Seção 3.3), apenas os campos de 1 a 6 são enviados na mensagem. As mensagens são descritas a seguir:

1. msg_{τ_r}: o tipo da mensagem, o qual pode incluir $INIT\_D$ (para configurar o sync-timer e estimar o diâmetro) ou $INIT\_S$ (para configurar o sync-timer e estimar a distância máxima), $SYNC$ (mensagem de sincronização), ACK (mensagem de reconhecimento) e $PULL$, a qual serve para recuperar nós isolados por meio do mecanismo de pulling proposto por QIU *et al.* (2018).
2. $msg_{src_{id}}$: identificador do nó emissor da mensagem.
3. $msg_{dest_{id}}$: identificador do nó de destino da mensagem, ou o valor $0xFFFF$, indicando que é uma mensagem de broadcast.
4. msg_{p_i}: identificador do nó pai do emissor da mensagem.
5. msg_{RTS}/msg_r: msg_{RTS} é o timestamp no momento de recebimento da mensagem $SYNC$ e é usado na mensagem do tipo $SYNC$. msg_r é o nó raiz do ciclo de ressincronização e é utilizado nas mensagens $INIT\_D$ ou $INIT\_S$.
6. msg_{STS}/msg_{c_r}: msg_{STS} é o timestamp no momento de envio de mensagem do tipo ACK. msg_{c_r} é o número ciclos de ressincronização.

7. msg_{h_r}: número de hops desde o nó raiz até o nó que recebeu a mensagem.

8. msg_{d_r}: estimativa do diâmetro da rede ou da distância máxima do nó raiz.

9. msg_{m_r}: valor máximo do nó raiz.

3.2 Modelagem de Relógio e Problema

Nesta seção nós descrevemos o modelo de relógio virtual adotado na Seção 3.2.1, e a modelagem do problema de seleção de nó raiz na Seção 3.2.2.

3.2.1 Modelo de Relógio Virtual

O oscilador do processador de um nó sensor emite um sinal elétrico em uma frequência real f_i para o nó i. Cada nó sensor i possui um registrador R_i que é incrementado por f_i^0, a frequência nominal, a qual é dada pelo fabricante. Dessa forma, o valor do relógio de hardware do nó i para o tempo absoluto t pode ser definido como $H_i(t) = \lfloor \frac{R_i(t)-R_i(t_0)}{f_i^0} \rfloor$, onde t_0 é o momento aleatório quando o relógio do nó foi inicializado e $R_i(t)$ é o momento de leitura do relógio de hardware no tempo t.

O nó sensor não conhece a frequência real f_i devido as condições de ambiente (temperatura, voltagem etc.), as quais alteram f_i, logo a frequência nominal aproxima a frequência real $f_i^0 \approx f_i$. De acordo com ELSON *et al.* (2003), o relógio local do nó sensor i pode aproximar ao tempo real t, por meio da função $H_i(t) = t + \theta_i$, onde θ_i é o offset absoluto do relógio do nó i. O offset absoluto é a diferença de tempo entre o relógio de um nó sensor em relação ao tempo real (absoluto). Se o tempo de referência é aquele fornecido pelo relógio de um nó sensor j, então o offset relativo θ_{ij} entre i e j é dado por $\theta_{ij}(t) = C_i(t) - C_j(t)$.

A Figura 3.1 exibe o modelo de sincronização de tempo aplicado por QIU *et al.* (2018), o qual é usado por nosso algoritmo de sincronização. Inicialmente, o nó i envia uma mensagem msg, com $msg_{RST} = C_i(T_1)$ e $msg_{type} = SYNC$ no momento T_1. No instante T_2, o nó j recebe a mensagem do tipo $SYNC$. O timestamp do nó j ao receber a mensagem $SYNC$ do nó i pode ser estimado como $C_j(T_2) = C_i(T_1) + \theta_{ij} + d$, onde d é o atraso de transmissão da mensagem. O nó j responde com uma mensagem do tipo ACK para o nó i no momento T_3, contendo $m_{RST} = C_j(T_2)$ e $msg_{STS} = C_j(T_3)$. O timestamp do nó i ao receber o ACK de j pode ser estimado como $C_i(T_4) = C_j(T_3) + \theta_{ij} + d$. Utilizando a abordagem SR e somando $C_j(T_2)$ com $C_i(T_4)$, o nó i pode estimar o offset para o nó i em T_4 como $\theta_{ij} = \frac{(C_j(T_2)-C_i(T_1))-(C_i(T_4)-C_j(T_3))}{2}$.

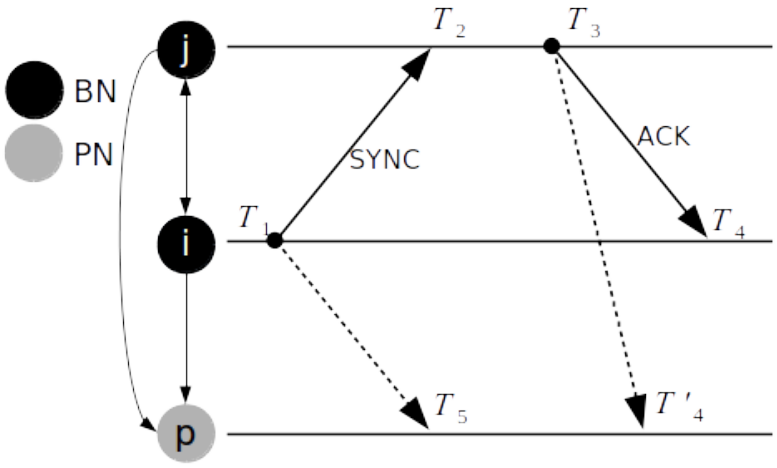


Figura 3.1: Modelo de sincronização de tempo do RE-Sync.

O nó p recebe o $SYNC$ no instante T5 e grava o tempo de recebimento $C_p(T_5)$ de acordo com seu relógio. Como o ACK contém o valor de $C_i(T_4)$ no campo m_{RST}, então o nó p pode estimar o offset para o nó i como $\theta_{pi} = C_p(T_5) - C_p(T_2) = C_p(T_5) - msg_{RST}$.

3.2.2 Modelagem do Problema

O problema de selecionar o nó raiz consiste em escolher o nó $r \in V$ que minimiza a distância de r para todos os outros nós de V. O conjunto de possíveis caminhos entre quaisquer dois nós v_0 e v_k, onde $0 \leq k \leq n$, é dado pela função $\pi(v_0, v_k) = \{\langle v_0, v_1, ..., v_k\rangle | \forall x \in \{1..n\} \Rightarrow (v_{x-1}, v_x) \in E\}$ e a distância total de um caminho $p \in \pi(v_0, v_k)$ é dada por

$$d(p) = \sum_{i=1}^{k} w(v_{i-1}, v_k) \tag{3.1}$$

A partir da Equação 3.1, $D(i, j)$ é definido como o custo do caminho mais curto entre dois nós i e j da seguinte maneira $D(i, j) = \min(\{d(p) \forall p \in \pi(i, j)\})$. Como o grafo G é não-direcionado, então $D(i, j) = D(j, i)$. O conjunto das distâncias entre um nó i e cada outro nó é definido como $M_i = \{D(i, j) \forall i \in V\}$. A distância do caminho mais longo entre o nó i e todos os outros nós é a distância do nó mais distante de i e é definida como $\max(M_i)$. O nó raiz r é aquele que minimiza as distâncias máximas entre todos os pares de nós, então r é o nó que minimiza $\max(M_i) \forall i \in V$. Então, o problema de encontrar o nó que minimiza as distâncias dele para todos os outros nós consiste em encontrar o nó r para o qual a seguinte expressão é verdadeira

$$\min(\max(M_r)) = \min_{\forall i \in V}(\max(M_i)) \tag{3.2}$$

3.3 Algoritmo de Sincronização de Relógios

Nesta seção, nós apresentamos o Robust and Efficient Clock Synchronization Algorithm (RE-Sync) para sincronização de relógios em RSSFs. O RE-Sync é composto de duas fases: a primeira é chamada de Fase de Seleção e a segunda é chamada de Fase de Sincronização. Na Fase de Seleção, os nós sensores realizam um processo de seleção do nó raiz e ao mesmo tempo sincronizam seus relógios. Na Fase de Sincronização, os nós sensores realizam apenas a sincronização de relógios.

Na Fase de Sincronização, o RE-Sync sincroniza os nós sensores por meio de trocas de mensagens two-way, combinando as metodologias SR e RR, e seguindo a abordagem de sincronização do RSync. No entanto, na Fase de Seleção, o RE-Sync utiliza essa troca de mensagens e um mecanismo de disparo de temporizador para selecionar o nó raiz da rede. O mecanismo de disparo de temporizador permite que o nó raiz seja selecionado utilizando apenas as mesmas mensagens do processo de sincronização de relógios, evitando overhead adicional de comunicação. As próximas seções apresentam as fases de seleção e de sincronização do RE-Sync.

3.3.1 Fase de Seleção

O objetivo do RE-Sync consiste em selecionar o nó raiz que minimiza a distância em hops do nó raiz para todos os outros nós da rede. No RE-Sync, cada nó sensor i possui duas variáveis principais d_i e m_i. A variável d_i contém a estimativa do nó i do diâmetro da rede e a variável m_i contém uma estimativa de $\max(M_i)$. O objetivo do RE-Sync é fazer com que o nó raiz seja o nó i que possua o menor valor de m_i, o que satisfaz a Equação 3.2. Além dessas variáveis, cada nó sensor possui as seguintes variáveis: r_i o nó raiz, c_i o número de ciclos (ressincronizações), h_i a distância em hops do nó i para o nó raiz r_i ($h_i = D(i, r_i)$). A Tabela 3.1 apresenta as variáveis utilizadas no RE-Sync.

A ideia principal do algoritmo na Fase de Seleção consiste de duas etapas: estimar o valor de $max_{\forall i \in 1..n} d_i$, o que é chamado de etapa de Estimativa de Diâmetro da Rede (EDR); e estimar o valor de $min_{\forall i \in 1..n} m_i$, o que é chamado de etapa de Estimativa da Distância Máxima (EDM). Durante a etapa EDR, o nó mais distante do nó raiz na rodada de re-sincronização c_i é escolhido como o nó raiz na rodada $c_i + 1$. Quando não ocorre mudança do valor estimado do diâmetro durante duas rodadas de re-sincronização consecutivas, então o RE-Sync inicia a etapa EDM. Durante esta etapa, o nó raiz é selecionado como aquele que possui o menor valor de m_i entre

Par.	Descrição
r_i	Nó raiz para o nó i
m_i	Distância máxima estimada de $M(i)$ para o nó i
d_i	Diâmetro estimado pelo nó i
c_i	Número de ciclos de ressincronização para o nó i
h_i	Número de hops de distância do nó i para o nó raiz r
m'_i	Variável auxiliar para guardar o valor de m_i
d'_i	Variável auxiliar para guardar o valor e d_i
t	Tipo de mensagem ($INIT\_D$ ou $INIT_S$
τ_i	Estado do nó sensor i

Tabela 3.1: Parâmetros do algoritmo de sincronização de tempo e descrições.

todos os nós da rede.

Para realizar a escolha do nó raiz nas duas etapas, cada nó sensor i faz uso de um temporizador $seltimer_i$. Durante a EDR, o $seltimer_i$ de cada nó i é configurado em função de $-h_i$, isto é $i \sim -h_i$. Dessa forma, o nó que possui maior valor de $-h_i$ sempre terá seu $seltimer_i$ disparado primeiro, sendo o próximo nó raiz.

Durante a etapa EDM, o $seltimer_i$ de cada nó i é configurado em função de m_i, isto é $i \sim m_i$. Dessa maneira, o nó com menor valor de m_i terá seu $seltimer_i$ disparado primeiro, sendo o próximo nó raiz.

A Figura 3.2 ilustra a ideia principal do algoritmo. Nesta figura, os nós sensores são representados como os vértices do grafo e dois nós vizinhos que estão na mesma área de cobertura são representados pelas arestas que ligam os vértices. O identificador do nó sensor corresponde ao número dentro do vértice e a função STD, a qual será detalhada na Seção 3.3.1, corresponde à função utilizada para configurar $seltimer_i$.

Inicialmente, cada nó i é configurado com $d_i = 0$, $h_i = 0$ e $m_i = 0$ e o nó raiz é escolhido aleatoriamente. Na Figura 3.2(a), o nó 6 é o nó escolhido como raiz. O nó raiz 6 envia uma mensagem msg para seus vizinhos (2, 3, 4, 8 e 9), os quais atualizam suas variáveis $d_i = msg_{d_r} = 0$ e $h_i = m_i = 1, \forall i \in \{2, 3, 4, 8, 9\}$ e configuram os seus seltimers como $seltimer_i = STD(msg_{d_r}, h_i)$. Inicialmente, o nó raiz não possui nenhuma informação para estimar o diâmetro da rede, logo o valor propagado por msg_{d_r} é igual a uma constante D.

A seguir na Figura 3.2(b), os nós 2, 3, 4, 8 e 9 enviam mensagens para seus vizinhos. Os nós 1, 5, 7, 10, 12 e 15 recebem as mensagens e atualizam suas variáveis $d_i = h_i = m_i = 2, \forall i \in \{1, 5, 7, 10, 12, 15\}$ e configuram cada $seltimer_i$ chamando $STD(msg_{d_r}, h_i)$. Esse processo continua até que o nó 13 configura $d_{13} = 4$, $h_{13} = 4$ e $m_{13} = 4$ como ilustrado na Figura 3.2(d). Como o temporizador do nó 13 foi configurado com o menor valor pela função STD, então $seltimer_{13}$ dispara primeiro que os demais e o nó 13 se torna o nó raiz na próxima rodada de re-sincronização

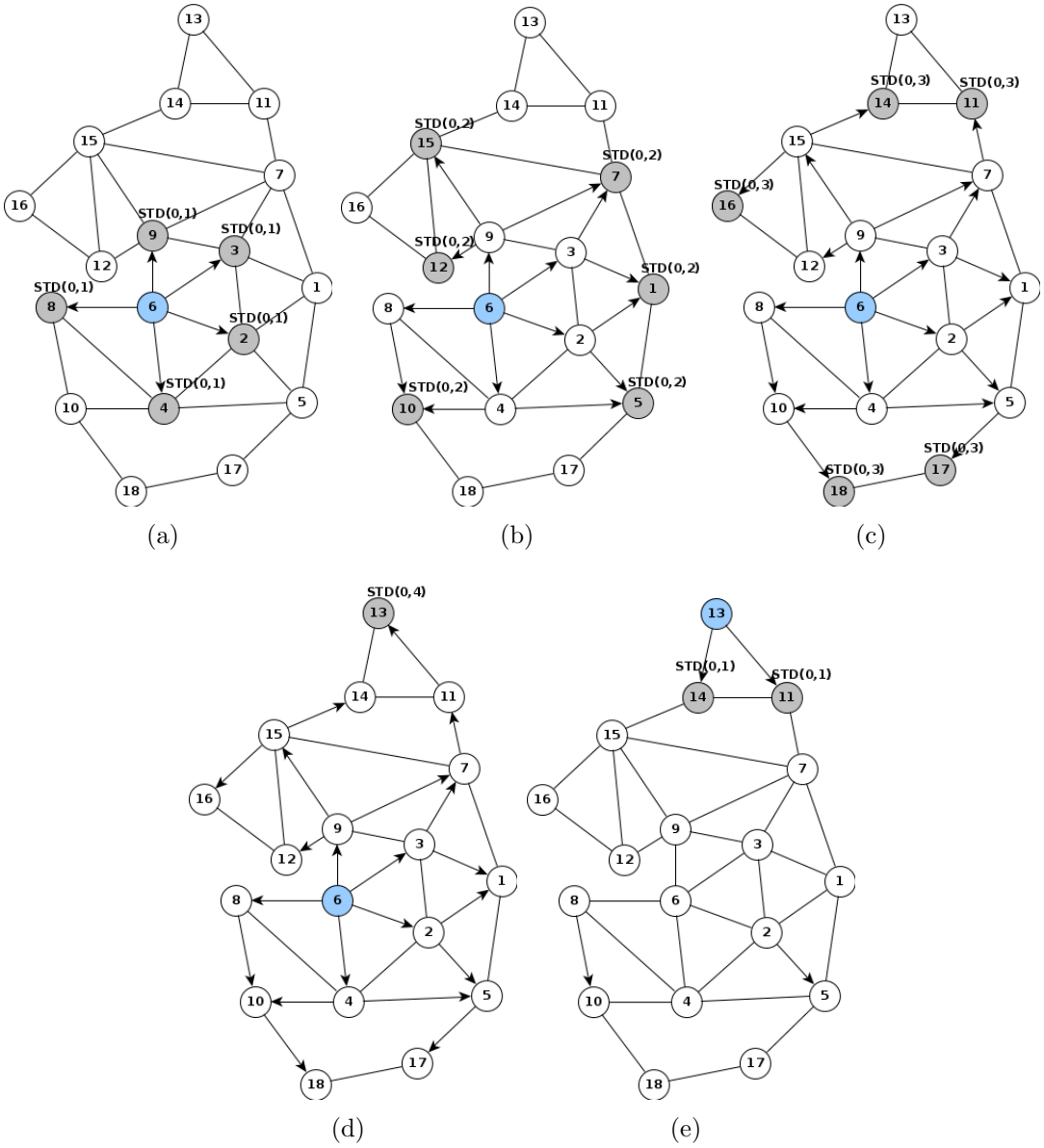


Figura 3.2: Troca de mensagens durante rodada inicial de sincronização para nó raiz 6 escolhido aleatoriamente. Cada nó sensor configura seu seltimer chamando $STD(d_i, h_i)$.

(Figura 3.2(e)). Então, o nó 13 envia uma mensagem para seus vizinhos e o processo se repete na rodada seguinte.

A Figura 3.3 mostra a árvore da primeira rodada de re-sincronização correspondente à Figura 3.2(a), já a Figura 3.3(b) mostra a árvore de sincronização para o nó raiz 13. Os nós 10 e 18 são os mais distantes do nó raiz, porém o nó 10 é escolhido como raiz da próxima rodada (Figura 3.3(c)). Nos casos em que dois ou mais são os mais distantes para o nó raiz, então o algoritmo seleciona aquele que possui o menor número de identificador como nó raiz. A Figura 3.3(c) mostra a árvore na terceira rodada de ressincronização, para o nó raiz 10. Na quarta rodada (Figura 3.3(d)), nem o valor de diâmetro, nem o valor máximo do nó raiz é alterado, então se inicia a etapa EDM. Para isso, cada nó i chama a função $STM(d_i, m_i, a)$ para configurar seu $seltimer_i$, onde a é uma constante para auxiliar a seleção do nó raiz. Quando existe mais de um candidato possível para ser o nó raiz, então aquele que possui o menor valor de número de identificador configura $a = 0$ e os demais configuram $a = \alpha$, onde $\alpha > 0$ é uma constante. A Tabela 3.2 apresenta os valores de r_i, d_i, m_i e τ_i considerando a execução do RE-Sync conforme a Figura 3.3.

Rod.	r_i	d_i	m_i	τ_i
0	6	0	0	$INIT\_D$
1	13	4	4	$INIT\_D$
2	10	6	6	$INIT\_D$
3	13	6	6	$INIT\_D$
4	1	6	3	$INIT\_S$
5	1	6	3	$INIT\_S$

Tabela 3.2: r_i, d_i e m_i para os nós raízes para as primeiras 6 rodadas da execução do RE-Sync.

Atualização de d_i e m_i

Ao receber uma mensagem msg de seu nó pai, o nó sensor atualiza sua distância (número de hops) para o nó raiz da seguinte forma

$$h_i = msg_{hop} + 1 \tag{3.3}$$

Então, o nó i atualiza d_i e m_i como descrito a seguir:

Atualização do diâmetro: O nó i atualiza sua estimativa do diâmetro da rede da seguinte forma

$$d_i = max(d_i, max(msg_{d_r}, h_i)) \tag{3.4}$$

A nova estimativa do diâmetro da rede d_i para o nó i é o maior valor entre o diâmetro estimado na rodada anterior de d_i, o diâmetro estimado pelo nó raiz msg_{d_r},

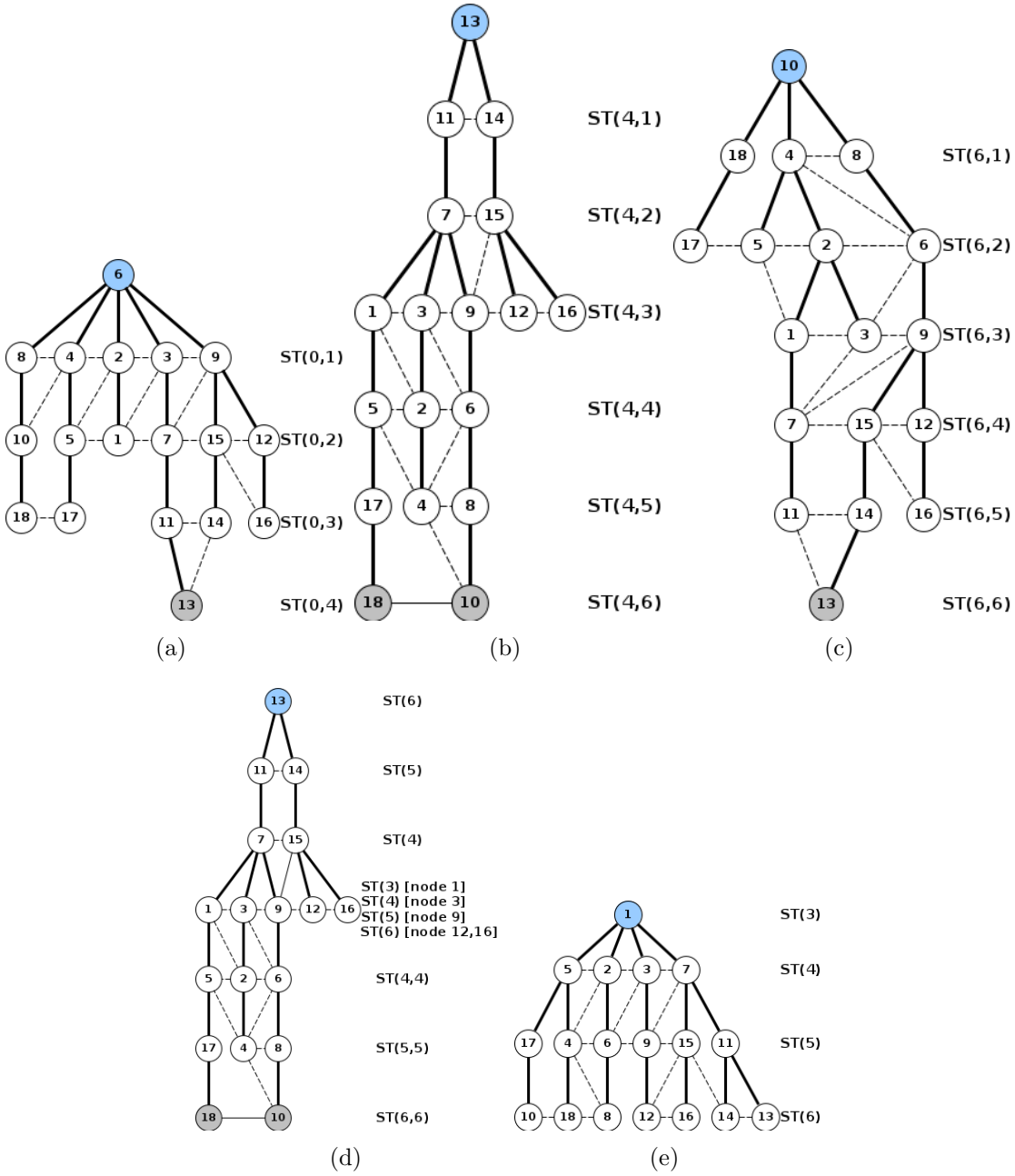


Figura 3.3: Árvores de sincronização geradas a partir da escolha dos nós raízes 6, 13 e 10 (Figura 3.2(a)-3.2(d)), durante a etapa de estimativa de diâmetro e escolha do nó 1 (Figura 3.2(e)) durante a etapa de estimativa de distância máxima.

propagado na mensagem msg, e a distância em hops h_i de i para o nó raiz.

Atualização da distância máxima: O nó i atualiza sua distância máxima da seguinte forma

$$m_i = max(m_i, h_i) \tag{3.5}$$

A nova estimativa da distância máxima de i corresponde ao maior valor entre m_i e a distância em hops h_i de i para o nó raiz.

Configuração de $seltimer_i$

A configuração do $seltimer_i$ pelo nó i depende de qual tipo de mensagem o nó i recebe de seu nó pai. Caso o nó i receba uma mensagem do tipo $INIT\_D$, então o nó i utiliza a função STD para configurar $seltimer_i$. Caso o nó i receba uma mensagem do tipo $INIT\_S$, então $seltimer_i$ é atualizado pela função STM.

A função STD é definida como

$$STD(d, h_i) = (I + 2 * d + 1 - h_i) * \beta \tag{3.6}$$

onde $0 \leq \beta \leq 1$ é uma constante e d corresponde à distância estimada do nó i para o nó raiz. A função STD retorna um valor que é proporcional ao valor de $-h_i$, de maneira que quanto maior o valor de h_i, menor é o valor de STD. Essa configuração é realizada para que o temporizador do nó mais distante do nó raiz seja disparado primeiro. O valor de d é multiplicado por 2 para evitar que a subtração resulte em um valor negativo. A função STM é definida como

$$STM(d, m_i, a) = (I + 2 * d + 1 + m_i + a) * \beta \tag{3.7}$$

onde $a \geq 0$ é o valor de atraso para auxiliar a seleção do nó raiz quando existem mais do que um candidato. O objetivo da função STM é fazer com que o $seltimer_i$ dispare mais rápido, quanto menor seja o valor de m_i. O valor a é usado para adicionar um atraso para alguns nós, no caso de mais do que um nó for escolhido como o mais distante do nó raiz.

O Algoritmo 1 apresenta o pseudo-código do RE-Sync e o Algoritmo 2 apresenta o pseudo-código para a função que descreve a Fase de Seleção do RE-Sync. No Algoritmo 1, inicialmente, o nó raiz escolhido inicia seu o processo de seleção disparando seu $seltimer_i$. As instruções condicionais nas linhas 6 e 9, evitam o conflito de dois nós raízes exemplificado na Figura 3.3(d).

Quando uma mensagem chega ao nó, se o número de ciclos do nó (c_i) e da mensagem (msg_{c_r}) é igual, significa que o nó i já recebeu uma mensagem de outro nó raiz durante aquele ciclo. Caso o número identificador do nó raiz originador da

Algorithm 1: RE-Sync

```
1  initialize m_i, d_i, c_i, h_i, m'_i, d'_i, t;
2  if i = r_i then
3  |  start seltimer_i to 0;
4  Upon receiving msg do
5  |  if msg_{τ_r} = INIT_D ∨ INIT_S then
6  |  |  if msg_{c_r} = c_i ∧ msg_r < r_i then
7  |  |  |  restore vars r_i, c_i, d_i, m_i, h_i;
8  |  |  |  cancel seltimer_i;
9  |  |  if msg_{c_r} > c_i then
10 |  |  |  save vars r_i, c_i, d_i, m_i, h_i;
11 |  |  |  Selection(msg);
12 |  else
13 |  |  Synchronization(msg);
14 Upon expiring seltimer_i do
15 |  SelTimer()
16 Upon expiring synctimer_i do
17 |  SyncTimer()
18 Upon expiring pulltimer_i do
19 |  PullTimer()
```

mensagem for menor do que o número identificador do nó raiz de i (r_i), então o nó i restaura o estado inicial de suas variáveis para os valores quando recebeu a mensagem do nó raiz r_i, além de cancelar seu $seltimer_i$.

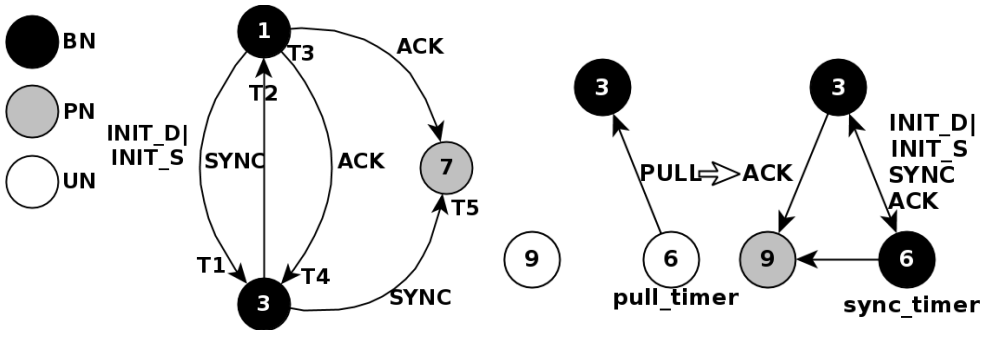
Em seguida, se o valor de msg_{c_r} é maior do que c_i, então o nó i salva o estado das variáveis e prossegue com a Fase de Seleção pela chamada da função *Selection*. Esse processo faz com que um nó propague apenas mensagens do nó raiz com o menor número de identificador, evitando o conflito de dois ou mais nós raízes iniciarem a sincronização no mesmo ciclo. Caso a mensagem recebida não seja do tipo $INIT\_D$ ou $INIT\_S$, então a Fase de Sincronização é iniciada pela chamada da função *Synchronization*.

3.3.2 Fase de Sincronização

A Fase de Sincronização ocorre em paralelo com a Fase de Seleção. Após o término da Fase de Seleção, então a Fase de Sincronização continua sincronizando os relógios isoladamente. Na Fase de Sincronização, um nó pode se tonar ou um nó BN ou um nó PN. Um nó BN sincroniza seu relógio com o relógio do nó pai por meio de troca de mensagens SR. Um nó PN sincroniza seu relógio sondando as mensagens trocadas entre dois nós BN por meio de sincronização RR. A Figura 3.4 mostra a sincronização de relógios SR e RR, além do funcionamento do mecanismo de pulling.

Algorithm 2: RE-Sync - Fase de Seleção

function *Selection (msg)*
  $d'_i \leftarrow d_i; m'_i \leftarrow m_i; \tau_i \leftarrow UN;$
  $p_i \leftarrow msg_{p_r}; c_i \leftarrow msg_{c_r}; r_i \leftarrow msg_r;$
  $h_i \leftarrow msg_{h_r} + 1;$
  $d_i \leftarrow \max(d_i, \max(msg_{d_r}, h_i));$
  $m_i \leftarrow \max(m_i, h_i);$
  $a = \begin{cases} \alpha, & m_i \geq m_r \wedge i > r \\ 0, & \text{otherwise} \end{cases}$
  if $msg_{\tau_r} = INIT\_D$ **then**
    if $msg_{d_r} \leq 0$ **then**
      $msg_{d_r} \leftarrow D_0;$
    start $seltimer_i$ **to** $STD(msg_{d_r}, h_i);$
    $\tau_i \leftarrow BN;$
  else
    if $msg_{\tau_r} = INIT\_S \wedge r_i \neq msg_r$ **then**
      if $d'_i \neq d_i \vee m'_i \neq m_i$ **then**
        start $seltimer_i$ **to** $STD(d_i, h_i);$
        $\tau_i \leftarrow BN;$
      else
        start $seltimer_i$ **to** $STM(d_i, m_i, a);$
  start $synctimer_i;$
function *STD(d, h)*
  return $[I + (2d + 1) - h] * \beta$
function *STM(d, m, a)*
  return $[I + (2d + 1) + m + a] * \beta$
function *SelTimer* ()
  $h_i \leftarrow 0; r_i \leftarrow i; c_i \leftarrow c_i + 1; \tau_i \leftarrow BN;$
  $t = \begin{cases} INIT\_D, & d'_i \neq d_i \vee m'_i \neq m_i \\ INIT\_S, & otherwise \end{cases};$
  if $\tau_i = INIT\_S \wedge d'_i = d_i \wedge m'_i = m_i$ **then**
    start $seltimer_i$ **to** $STS(d_i, m_i, 0);$
  broadcast $(< t, i, 0xFFFF, p_i, i, c_i,$
    $h_i, d_i, m_i >);$



(a) (b)

Figura 3.4: Esquema de sincronização de relógios (a) e mecanismo de pulling (b).

Na sincronização SR da Figura 3.4(a), entre os nós BN 1 e 3, inicialmente 1 envia uma mensagem de broadcast $INIT\_D$ ou $INIT\_S$ que é recebida por 3 e 7. O nó 3 configura seu $synctimer_3$, o qual é disparado e, então, o nó 3 envia uma mensagem $SYNC$ para 1 contendo o tempo de envio da mensagem $SYNC$ T1. O nó 7 recebe a mensagem e armazena o tempo de chegada T5. O nó 1 recebe o $SYNC$, armazena seu tempo de chegada T2 e responde ou com uma mensagem ou ACK, contendo T2 e o tempo de envio T3, para o nó 3. O nó 3 recebe o ACK no tempo T4 e com os quatro timestamps sincroniza seu relógio. O nó 7 escuta as mensagens $SYNC$ de 3 para 1 e ACK de 1 para 3, obtendo os timestamps T2, embutido na mensagem ACK de 1 para 3, e T5. Com os valores de T2 e T5, o nó 7 é capaz de sincronizar seu relógio.

Quando um nó UN não recebe uma mensagem $INIT\_D$ ou $INIT\_S$, depois de algum tempo seu *pull_timer* é disparado e então, o nó envia uma mensagem $PULL$. Na Figura 3.4(b) o nó 6 envia uma mensagem $PULL$ para o nó 3. Caso o nó 3 seja um nó UN, ele se torna um nó BN e inicia a sincronização SR com o nó 6. Na Figura 3.4(b), o nó 9 realizou sincronização RR, por meio da sondagem das mensagens trocadas 6 tre o nó 3 e o nó 6.

- **Esquema de sincronização de relógios**

  A seguir nós apresentamos uma descrição detalhada da Fase de Sincronização, exemplificada na árvore de sincronização da Figura 3.3(e).

  Inicialmente o nó 1 envia uma mensagem $INIT\_D$ ou $INIT\_S$ de broadcast para seus vizinhos. Para cada nó UN que recebe uma mensagem $INIT\_D$ ou $INIT\_S$ (i.e. nós 5, 2, 3, 7) temos que:

  – Se $pulltimer_i$ do nó está configurado, então ele é cancelado.

  – Senão, $\forall i \in \{5, 2, 3, 7\}$ $p_i = m_{src_{id}}$ e configura o $synctimer_i$. Quando $synctimer_i$ expira, o tipo de nó é alterado de UN para BN. Então, cada nó i (5, 2, 3, 7) envia uma mensagem $SYNC$ contendo o tempo local T1.

Para o nó que recebe uma mensagem m do tipo $SYNC$:

- Se o nó é UN, ele grava o timestamp local T5 no recebimento da mensagem e cancela seu temporizador $synctimer_i$. Então ele muda seu tipo de nó de UN para PN.
- Se $m_{dest_{id}} = i$, então o nó i grava o timestamp local T2 ao receber a mensagem e responde com uma mensagem ACK no momento T3. A mensagem ACK carrega os timestamps T2 e T3 nos campos m_{RTS} e m_{STS}.

Para o nó que recebe uma mensagem ACK:

- Se $m_{dest_{id}} = i$, então o timestamp local T4 do momento de chegada do ACK é gravado. O offset pode ser calculado pelos quatro timestamps T1, T4, m_{RTS} e m_{STS} e o nó i pode sincronizar seu relógio. Então o nó i envia uma mensagem de broadcast $INIT\_D$ ou $INIT\_S$ para seus vizinhos.
- Senão, o nó i torna-se PN e pode usar o valor de m_{STS} e o timestamp T5 para calcular o offset e sincronizar com seu nó pai por meio de sincronização RR.

- **Configuração de $synctimer_i$ e $pulltimer_i$**

  O valor do $pulltimer_i$ é configurado em função da distância h_i do nó i para o nó raiz e o tempo médio de sincronização em cada hop como descrito por QIU *et al.* (2018). Dessa forma, o valor de $pulltimer_i$ é configurado como $h_i * \delta$, onde δ é o tempo médio de sincronização em cada hop durante a simulação. Em QIU *et al.* (2018), o tempo de inicialização da sincronização é somado a $h_i * \delta$, porém, como o RE-Sync não necessita realizar uma inicialização do processo de sincronização, nós não utilizamos este termo na equação, pois seu valor seria igual a 0. O $pulltimer_i$ é configurado de acordo com a distância para o nó raiz para evitar que o $pulltimer_i$ seja disparado múltiplas vezes. O valor do $synctimer_i$ é definido aleatoriamente da seguinte forma $synctimer_i = X_i \varpi$, onde $0 \leq X_i \leq 1$ é uma variável aleatória uniforme e ϖ é o tempo de espera médio para o $synctimer_i$ ser disparado.

O Algoritmo 3 descreve em forma de pseudo-código o processo executado pelo RE-Sync durante a Fase de Sincronização. A função *Synchronization* é responsável por tratar as mensagens $SYNC$ e ACK e sincronizar os relógios dos nós sensores. A função *SyncTimer* dispara o $synctimer_i$ do i, para que o nó envie uma mensagem $SYNC$ e realize a sincronização com seu nó pai. A função *PullTimer* envia uma mensagem do tipo $PULL$ e é disparada quando um nó não é sincronizado na rede.

Algorithm 3: RE-Sync - Fase de Sincronização

```
1  function Synchronization (msg)
2      if msg_type = SYNC then
3          T2 ← ReadClock();
4          if msg_dst_id ≠ i then
5              T5 ← T2;
6              if τ_i = UN then
7                  τ_i = PN;
8                  Cancel(synctimer_i)
9          if msg_dst_id = i then
10             T3 ← ReadClock();
11             broadcast (< ACK, i, msg_src_id, p_i, T2, T3,
12                          h_i, d_i, m_i >);
13     if msg_type = ACK then
14         if msg_dst_id = i and msg_src_id = p_i then
15             T4 ← ReadClock();
16             θ ← [(msg_RTS − T1) − (T4 − msg_STS)]/2;
17             AdjustClock(θ);
18             broadcast (< t, i, 0xFFFF, p_i, −1, −1,
19                          h_i, d_i, m_i >);
20         else
21             if τ_i = PN and msg_src_id = p_i then
22                 θ ← msg_RST − T5;
23                 AdjustClock(θ);
24 function SyncTimer ()
25     τ_i ← BN;
26     T1 ← ReadClock ();
27     broadcast (< SYNC, i, 0xFFFF, p_i, −1, −1,
28                  h_i, d_i, m_i >);
29 function PullTimer ()
30     broadcast (< PULL, i, 0xFFFF, p_i, −1, −1,
31                  h_i, d_i, m_i >);
```

Capítulo 4

Avaliação do Algoritmo de Sincronização de Tempo

Neste capítulo, nós apresentamos a avaliação experimental do algoritmo de sincronização de tempo. Na Seção 4.1, nós apresentamos a metodologia de avaliação. Na Seção 4.2 nós apresentamos a configuração e principais parâmetros dos experimentos. Finalmente na Seção 4.3, nós apresentamos os resultados obtidos.

4.1 Metodologia de Avaliação

Nós realizamos uma avaliação experimental com o objetivo de verificar a eficiência do RE-Sync em manter a acurácia dos relógios dos nós sensores da rede e evitar o aumento do número de troca de mensagens de sincronização (overhead de comunicação).

As métricas avaliadas nos experimentos foram Erro de Sincronização Global (ESG), Erro de Sincronização Local (ESL), Número de Broadcast (NB) e Porcentagem de Nós Sincronizados (PNS). O ESG corresponde à diferença máxima entre os valores de leitura de relógio de todos os nós sensores da rede e o valor de relógio do nó raiz em um mesmo instante de tempo. O ESL é a diferença máxima entre o valor de relógio de um nó e o valor de relógio dos seus vizinhos em um mesmo instante de tempo. O NB é o número total da soma de mensagens de sincronização enviadas por cada nó sensor da rede. O PNS corresponde à porcentagem de nós que mudaram seu tipo inicial de UN para BN ou PN.

Nós executamos um conjunto de experimentos e comparamos o desempenho do RE-Sync com o RSync proposto por QIU *et al.* (2018) para diversos cenários. O RSync é um recente procotolo de sincronização eficiente em consumo de energia. Na Seção 4.2 nós descrevemos a configuração do ambiente experimental e na Seção 4.3 nós mostramos os resultados obtidos.

4.2 Configuração Experimental

Para realizar os experimentos, nós utilizamos o simulador de eventos discretos para RSSFs chamado sinalgo, o qual é escrito na linguagem de programação Java e pode ser acessado em SINALGO (2020). Nós configuramos o simulador para posicionar aleatoriamente cada nó sensor em uma área quadrada 1000 x 1000 m até que todos os nós estivessem conectados. O nó raiz foi configurado para iniciar o processo de ressincronização periodicamente no intervalo de 10 s (QIU *et al.* (2018)).

A fim de reduzir o impacto de erros aleatórios, nós coletamos 10000 amostras de tempo de cada rodada de experimento, além disso, cada experimento foi repetido 30 vezes. Nós configuramos o atraso de transmissão no simulador como uma distribuição gaussiana com média de 150 ms (LENZEN *et al.* (2015)). Nós variamos o número de nós com o seguinte conjunto de valores 200, 250, 300, 350, 400, 450. Nós também variamos o raio de alcance do sinal de rádio dos nós sensores, em metros, com valores do conjunto 85, 100, 115, 130, 145, 160.

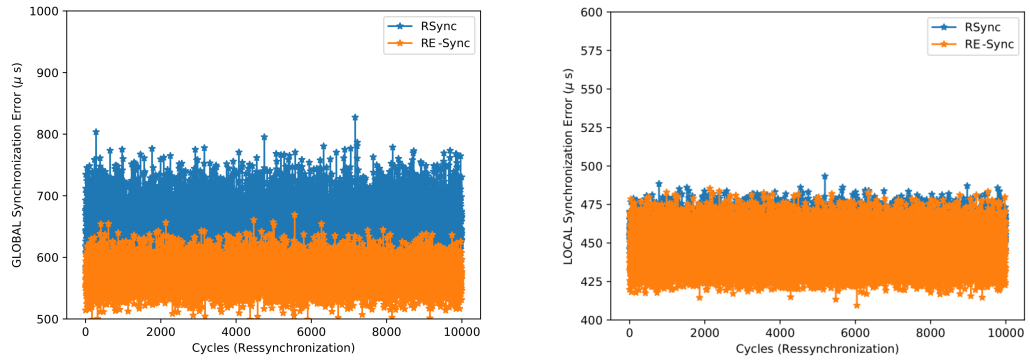
4.3 Resultados Experimentais

Nesta seção, nós apresentamos os resultados obtidos das métricas ESG e ESL variando o tempo, além dos resultados de ESG variando a distância em hops para o nó raiz. Nós também mostramos os resultados do NB variando a quantidade de nós sensores na rede e a distância do raio de alcance do sinal de rádio dos nós sensores. Finalmente, nós apresentamos os resultados da quantidade de nós sincronizados variando a quantidade e nós sensores da rede.

4.3.1 Erro de Sincronização

A fim de comparar o ESG e o ESL entre o RE-Sync e o RSync, nós executamos um experimento com 10000 ciclos de ressincronização e a Figura 4.1 mostra os resultados. A Figura 4.1(a) mostra que o ESG do RE-Sync foi menor do que o erro de sincronização do RSync. A média do ESG para o RE-Sync foi de 568 μs e para o RSync foi de 671 μs, já o valor máximo do ESG para o RE-Sync foi de 668 μs e para o RSync foi de 826 μs. Dessa forma, a melhoria média do RE-Sync com relação ao RSync foi de 15.35 % e a melhoria do valor máximo foi de 19.12 %. Essa melhoria ocorreu pois o RE-Sync seleciona o nó raiz que minimiza a sua distância em hops em relação a todos os outros nós da rede. Como o erro de sincronização aumenta diretamente proporcional à distância em hops, o RE-Sync reduz o erro de sincronização entre os relógios dos nós quando comparado com o RSync.

A Figura 4.1(b) mostra o ESL para o RE-Sync e RSync durante 10000 ciclos de ressincronização. Com relação ao ESL, os dois algoritmos não apresentam diferença



(a) (b)

Figura 4.1: Erro de sincronização global (a) e erro de sincronização local (b), variando a quantidade de nós sensores.

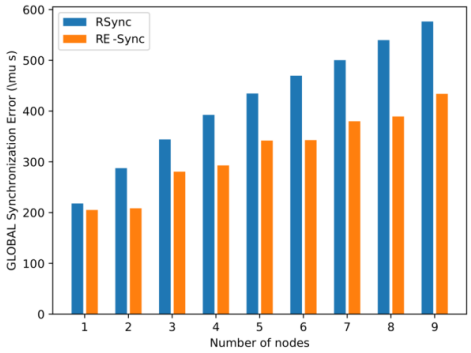
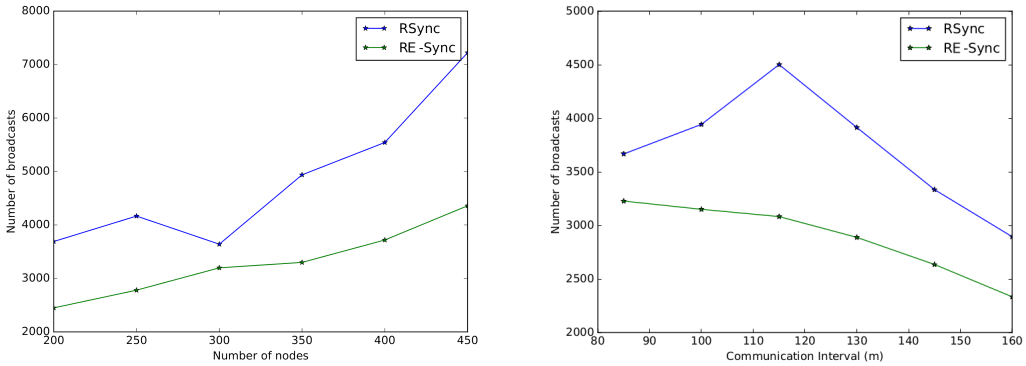


Figura 4.2: Erro de sincronização variando a distância em hops para o nó raiz.

significativa de desempenho. O valor médio do ESL para o RE-Sync e RSync foi de cerca de 448 μs e 451 μs, respectivamente. O valor máximo do ESL para o RE-Sync foi de 485 μs e para o RSync foi de 493 μs. Não há diferença significativa do ESL para os dois algoritmos, pois o cálculo do ESL é computado para um nó em relação a seus vizinhos, os quais sempre estão a 1 hop de distância um do outro. Como o RE-Sync reduz a distância de um nó qualquer para o nó raiz, ele produz um impacto de redução do erro global da rede, ou seja, do ESG, e não um impacto local.

A Figura 4.2 apresenta o ESG no eixo-y e o número de identificador do nó sensor no eixo-x. Para realizar este experimento, nós posicionamos os nós em uma topologia em linha de modo que o número do identificador corresponde à distância em hops do nó para o nó raiz. A Figura 4.2 mostra que o ESG aumenta em função da distância para o nó raiz. Esse aumento ocorre devido ao problema do erro cumulativo do protocolo de sincronização. O RE-Sync supera o RSync pois ele atenua o problema do erro cumulativo e reduz a distância em hops da maioria dos nós para o nó raiz.



(a) (b)

Figura 4.3: Número de mensagens de broadcasts enviadas pelos nós sensores, variando a quantidade de nós sensores (a) e o raio de alcance do sinal de rádio dos nós sensores(b).

4.3.2 Número de broadcasts

Nós comparamos a soma do número de broadcasts (NB) realizados pelos nós sensores para o RE-Sync e o RSync. Na Figura 4.3(a) nós avaliamos o NB variando o número de nós sensores. Nesta figura, nota-se que o NB aumentou proporcionalmente ao número de nós sensores na rede. Além disso, o desempenho do RE-Sync foi maior do que o desempenho do RSync. O NB variou de 2422 até 4433 broadcasts para o RE-Sync e o NB variou de 3674 até 7213 broadcasts para o RSync.

A explicação para o menor número de broadcasts do RE-Sync comparado com o RSync, é que na Fase de Seleção, o intervalo de ressincronização muda de acordo com o nó sensor que foi escolhido como nó raiz. Este valor é maior do que o intervalo de ressincronização durante o restante da operação do algoritmo na Fase de Sincronização. Isso faz com que o número de mensagens de broadcast seja menor durante a Fase de Seleção, do que o restante de período de operação do algoritmo, impactando o valor da métrica NB. Nós também observamos que o RE-Sync gera uma árvore de sincronização mais balanceada do que o RSync. O RE-Sync seleciona um nó próximo ao centro do grafo, enquanto o RSync seleciona o nó raiz aleatoriamente com relação a sua posição topológica. Dessa forma, com o RE-Sync, os nós próximos ao nó raiz, incluindo ele próprio, tendem a sincronizar mais nós vizinhos em uma mesma rodada de ressincronização, do que com o RSync.

Na Figura 4.3(b), nós avaliamos o NB variando o raio de alcance do sinal de rádio dos nós sensores. Nota-se que o NB diminui conforme se aumenta o alcance dos nós sensores. Aumentar o alcance do sinal de rádio faz com que os nós sensores possuam mais nós vizinhos, aumentando a quantidade de nós PN na rede. Como os nós PN não realizam broadcasts, o NB é reduzido. Na Figura 4.3(b), o NB para o RE-Sync é menor do que o NB para o RSync.

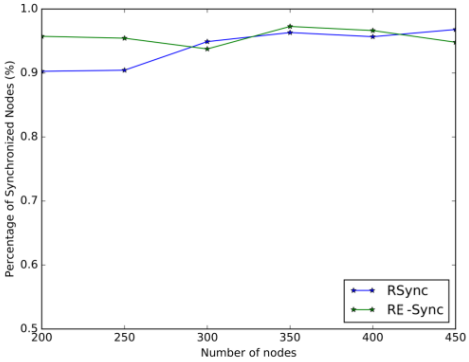


Figura 4.4: Porcentagem de nós não-sincronizados variando a quantidade de nós sensores.

Uma hipótese para explicar a diferença de desempenho é que a escolha do nó raiz do RE-Sync faz com que os nós tenham mais vizinhos, principalmente no centro do grafo da topologia. Neste cenário, nó raiz tende a estar próximo ao centro, e não na periferia, do grafo da rede. Dessa forma, um único broadcast de um nó pai pode sincronizar mais nós filhos ao mesmo tempo na árvore de sincronização gerada pelo RE-Sync do que na árvore gerada pelo RSync.

4.3.3 Porcentagem de Nós Sincronizados

A Figura 4.4 mostra o número de nós sensores no eixo-x e a porcentagem de nós sincronizados no eixo-y. O protocolo RE-Sync é capaz de sincronizar aproximadamente 95% dos nós em todos os cenários, enquanto o protocolo RSync apresenta uma porcentagem de nós sincronizados de 90% até 250 nós e 95% aproximadamente a partir de 300 nós.

O principal motivo para o RSync não atingir o valor de 95% de nós sincronizados para os cenários com 200 e 250 nós sensores é que neste algoritmo alguns nós que deveriam se tornar do tipo PN permanecem como nós do tipo UN durante a sincronização. Um nó se torna PN quando ele escuta do meio broadcast uma mensagem $SYNC$ seguida por uma mensagem ACK, nesta ordem, de dois nós diferentes. No entanto, devido ao atraso aleatório proveniente da transmissão da mensagem pelo link de comunicação , existe a possibilidade do nó receber uma mensagem ACK antes de uma mensagem $SYNC$. Neste caso, o nó se confunde e não muda seu tipo para PN.

O RE-Sync apresenta um desempenho levemente superior ao RSync para 200 e 250 nós. A partir de 300 nós o RSync consegue apresentar desempenho semelhante ao RE-Sync pois com o aumento de nós sensores, se aumenta a densidade da rede e, por consequência, a quantidade de vizinhos com que um nó sensor pode se sincronizar.

Capítulo 5

Alocação de Recursos na Nuvem das Coisas

Este capítulo apresenta nossa proposta de alocação de recursos para ecossistemas da nuvem das coisas. Nosso algoritmo faz uso de virtualização, a qual permite desacoplar a infraestrutura física das aplicações. O modelo de virtualização proposto faz uso do conceito de tipo de dado, o qual associa cada nó virtual com o dispositivo da IoT que possui o mesmo tipo de dado. Além disso, o nó virtual pode expandir ou retrair seus recursos para se adaptar as restrições da requisição. O modelo de virtualização é baseado em agentes econômicos e fornece um valor de utilidade para o nó virtual, além de precificar as requisições das aplicações. Por meio dessa abordagem, o algoritmo de alocação de recursos é capaz de distribuir a carga de trabalho e tratar as requisições respeitando suas prioridades. O algoritmo de alocação de recursos também considera uma cache presente em cada nó de borda, para armazenamento do dado da requisição. O atendimento de requisições com dados provenientes da cache evita que o nó de borda colete dados da camada das coisas desnecessariamente, aumentando o tempo de vida dos dispositivos da IoT. Nosso algoritmo também trata o problema do frescor de dado, buscando o dado do dispositivo da IoT apenas quando o valor de frescor corresponde ao tempo imediato.

Na Seção 5.1, nós apresentamos a proposta de arquitetura do nosso sistema para o ambiente da CoT. Na Seção 5.2 nós definimos os componentes da arquitetura e os parâmetros do algoritmo de alocação de recursos. Na Seção 5.3, nós apresentamos o algoritmo que organiza a infraestrutura da CoT. Na Seção 5.4, nós apresentamos nosso algoritmo de alocação de recursos e o modelo de nós virtuais baseado em mercado.

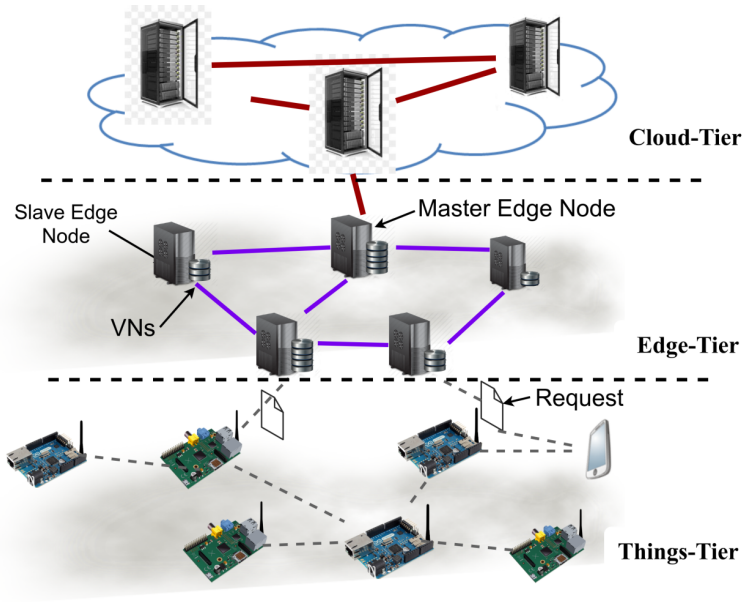


Figura 5.1: Arquitetura da CoT de três camadas.

5.1 Arquitetura do Sistema

Nós consideramos um ecossistema da IoT composto de três camadas arquiteturais: camada das coisas (Things Tier), camada de borda (Edge Tier) e camada da nuvem (Cloud Tier), como descrito por LI *et al.* (2017). A organização da camada de borda considerada em nossa proposta é hierárquica e possui dois tipos de nós, nós mestres e nós escravos, em que nós mestres são responsáveis por gerenciar o processo de alocação. A Figura 5.1 mostra os elementos da arquitetura de três camadas, os quais compõem o ecossistema da CoT em nosso trabalho. No topo da arquitetura está a camada da nuvem, a qual é composta por datacenters que possuem grande quantidade de recursos para atender as requisições das aplicações, e é distante da fonte de dados. A camada de borda é composta por típicos dispositivos físicos de borda, os quais variam desde gateways, roteadores inteligentes, micro-clusters, até estações de rádio-base. A proximidade dos nós de borda com as entidades físicas, na camada das coisas, habilita um acesso mais rápido para a fonte de dados. A camada de borda é baseada em virtualização e os dispositivos dela podem fornecer recursos de processamento e armazenamento para serviços sendo executados na borda da rede. A camada das coisas é a camada inferior, a qual é responsável por conectar as entidades físicas com a Internet, por meio dos dispositivos da borda. A entidade física é qualquer objeto físico de interesse com o objetivo de sensoriamento ou atuação. Entidades físicas integram recursos de processamento e comunicação sem fio, e elas são instrumentadas com sensores e atuadores.

Na organização hierárquica proposta é necessário definir quais nós de borda são os nós mestres. Para isso, nós adotamos dois critérios: os nós de borda que possuem

maior quantidade de recursos; e os nós de borda que minimizam a distância para os demais nós de borda da rede. Para atender ao primeiro critério, os nós escolhidos como nós mestres são aqueles que possuem maior capacidade computacional e maior capacidade de comunicação com a nuvem. A capacidade computacional consiste da quantidade de recursos que o nó possui em termos de taxa de processamento, quantidade de memória de acesso aleatório (RAM - Ramdom-Access Memory) e quantidade de armazenamento permanente no disco rígido. A capacidade de comunicação corresponde à largura de banda dos links de comunicação do nó.

O segundo critério define a posição lógica (um nível de hierarquia) de cada um dos nós escravos da camada de borda. O nível de hierarquia denota um valor de ordem de um nó com relação ao nó mestre e é definido a partir da distância geográfica física entre os nós. O nó mestre está sempre no primeiro nível e seus respectivos nós escravos nos demais níveis. Essa hierarquia determina a comunicação dos nós, por exemplo, nós no segundo nível se comunicam com o nó mestre ou nós escravos do terceiro nível. O estabelecimento da hierarquia dos nós, considerando a distância geográfica entre eles, visa levar em conta a questão do atraso de propagação para longas distâncias.

Nós consideramos cenários de aplicações em que os nós de borda estão dispostos em grandes áreas geográficas, áreas urbanas (por exemplo, uma aplicação smart city), ambientes agrícolas, campi universitários, grandes e complexas construções civis (parte de uma corporação ou empresa) etc., e que a distância entre dois nós de borda é da ordem de quilômetros. Por exemplo, ALVAREZ-CAMPANA *et al.* (2017) e WEBB e HUME (2018) são propostas para monitorar um campus universitário com áreas de 5,5 km^2 e 9,68 km^2, respectivamente. Nesse tipo de cenário, a densidade espacial de nós de borda poderia ser aumentada para evitar o atraso de propagação para longas distâncias. No entanto, o custo de aquisição e manutenção de equipamento para aumentar a densidade de nós de borda também é muito maior. Logo, nós assumimos cenários em que as distâncias entre dois nós de borda podem atingir a ordem de quilômetros de distância.

Um conceito básico necessário para entender o algoritmo de alocação de recursos proposto é o tipo de dado. Os usuários definem suas aplicações com base nos tipos de dados disponíveis na infraestrutura, e o provedor de infraestrutura atende as requisições dos usuários também com base nos tipos de dados. Um tipo de dado pode representar um dado bruto (um valor de temperatura) adquirido por um sensor, uma série temporal de dados detectados, ou mesmo dados resultantes de algum processamento (um evento ou uma agregação de dados). Diferentes tipos de dispositivos da IoT, por exemplo, nós de sensores ou uma câmera, podem produzir diferentes tipos de dados. Na definição de um tipo de dado, além da informação específica sobre o tipo da variável de ambiente, também é incluída a quantidade de

dados que deve ser lida para definir o tipo de dado. Finalmente, um tipo de dado também pode especificar uma função de fusão de informações que transforma as entradas em dados de saída. Por exemplo, um tipo de dado pode ser uma sequência de dez leituras de valores brutos de temperatura obtidos em uma posição específica em uma sala de um edifício com uma determinada taxa de coleta. Como outro exemplo, um tipo de dado pode ser um conjunto de imagens processadas, por meio de uma função específica, de câmeras de monitoramento em um estacionamento em uma determinada taxa de atualização.

O algoritmo de alocação de recursos proposto conecta os nós de borda e os dispositivos da IoT com as aplicações, usando o conceito de tipo de dado. Em nossa proposta, a camada de borda é virtualizada usando o conceito de nó virtual (VN - Virtual Node). O VN é um componente de software instanciado para fornecer dados de um único tipo de dado. A definição do VN (Seção 5.4.1) foi herdada do modelo de virtualização para uma arquitetura de nuvem das coisas proposto por SANTOS *et al.* (2015). Nosso conceito de VN modifica o modelo de VN dos autores, definindo vários tipos de VN. Cada tipo de VN tem uma responsabilidade definida dentro do sistema e corresponde a um tipo de dado específico. Dessa forma, o tipo de dado é usado para descrever os dados a serem fornecidos por um tipo específico de VN. O VN implementa uma função de fusão de informações e deve incluir todos os atributos do tipo de dado que ele fornece.

O principal objetivo da infraestrutura da CoT é reduzir a latência de atendimento das requisições das aplicações, logo o algoritmo de alocação de recursos visa atender a maior quantidade de requisições na camada de borda. Nosso algoritmo de alocação de recursos também permite que nós mestres colaborem entre si encaminhando requisições uns para os outros, caso estejam conectados. Dessa forma, o nó mestre pode envolver outros nós de borda mestre vizinhos para encaminhar uma requisição, caso não tenha sido capaz de alocar a requisição da aplicação a nenhum dos seus nós escravos. Este processo de colaboração permite que toda a camada de borda seja envolvida para o atendimento da requisição, promovendo um melhor compartilhamento de recursos. A seguir, nós apresentamos o modelo de aplicação da arquitetura proposta.

5.1.1 Modelo de Aplicação

Uma aplicação submetida para ser atendida pelo sistema é composta de um conjunto de requisições para acessar dados de um dispositivo com um determinado tipo de dado. A aplicação é executada no dispositivo da IoT e gera requisições que são encaminhadas para o nó de borda conectado a este dispositivo. A requisição é processada pelo nó de borda conectado ao dispositivo da IoT que a gerou, o qual é

denominado nó de borda origem, em algum outro nó de borda, ou pela nuvem. O resultado do processamento da carga de trabalho da aplicação é retornado para o nó de borda origem, o qual encaminha o resultado da requisição de volta ao dispositivo da IoT que executa a aplicação. O caminho crítico de uma requisição é descrito nas etapas abaixo:

1. A requisição é gerada pela aplicação que é executada no dispositivo da IoT.
2. A aplicação do dispositivo da IoT envia a requisição para o nó de borda origem conectado a ele.
3. O nó de borda origem processa a requisição ou a encaminha para algum nó de borda vizinho, ou para a nuvem.
4. Os nós de borda encaminham a requisição entre si até que algum nó de borda processa a requisição ou ela é encaminhada para a nuvem.
5. Algum nó de borda, ou a nuvem, processa a requisição.
6. O nó de borda, ou a nuvem, que processou a requisição encaminha-a de volta para o nó de borda origem.
7. O nó de borda origem encaminha a requisição para o dispositivo da IoT.

De acordo com o caminho crítico descrito acima, cada requisição pode chegar ao nó de borda por um dos dispositivos da IoT conectados a ele (camada das coisas) ou por algum outro nó de borda (camada de borda). Quando um nó de borda não possui recursos suficientes para atender uma requisição, ele pode encaminhá-la para ser atendida por algum de seus vizinhos ou para a nuvem. Nós assumimos que um algoritmo de roteamento garante que a resposta da requisição será retornada ao nó de borda origem. Além disso, nós propomos um algoritmo de infraestrutura que especifica uma organização hierárquica dos nós de borda, fornecendo uma especificação das rotas por onde as requisições podem ser encaminhadas.

As requisições são independentes e os dados para atender as requisições são provenientes dos VNs. O algoritmo de alocação de recursos, portanto, aloca VNs para atender às requisições das aplicações. Tais VNs estão em algum nó de borda que está conectado à um dispositivo da IoT (um nó sensor, por exemplo) que gera os dados requisitados pela aplicação.

Uma requisição se refere ao dado produzido por um dispositivo da IoT e ao processamento daquele dado. Nós assumimos que o dado produzido são conjuntos de medidas temperatura dos nós sensores. É assumido que o VN que coleta o dado do dispositivo da IoT para atender a requisição é aquele que também processa o dado coletado. Logo, o dado solicitado pela requisição da aplicação é coletado e processado pelo mesmo VN, no mesmo nó de borda.

5.2 Definições Preliminares e Parâmetros

A camada de borda é representada por um grafo $G = (E, L, d)$, no qual $E = \{e_i | \forall i \in 0..n\}$ é o conjunto de n nós de borda, $L = \{(e_i, e_j) \forall i, j \in 1..n \wedge e_i, e_j \in E\}$ é o conjunto de links de comunicação entre nós de borda e d é a função de distância euclidiana entre dois nós de borda $d : E \times E \rightarrow \mathbb{R}$.

Um nó de borda e_i é definido como uma tupla $e_i = (C_{e_i}, M_{e_i}, S_{e_i}, \tau_{e_i}, D_{e_i}, B_{e_i}, Q_{e_i})$ para $i > 0$, em que C_{e_i} é a taxa de processamento, M_{e_i} é a quantidade total de memória, S_{e_i} é a quantidade total de armazenamento do disco rígido, $\tau_{e_i} = \{MASTER, SLAVE\}$ é o tipo do nó de borda, D_{e_i} é o conjunto de tipos de dados que podem ser fornecidos pelo nó de borda e_i, B_{e_i} é a cache local de e_i em que o nó de borda armazena dados de sensoriamento coletados da camada das coisas, e Q_{e_i} é a fila de requisições em que o nó de borda armazena as requisições que não foram atendidas imediatamente.

A cache local é definida como o conjunto $B_{e_i} = \{(t_{ib}, \delta_{ib}) | t_{ib} \in Q^+ \wedge \delta_{ib} \in D_{e_i}\}$, para $b > 0$, em que t_{ib} é o instante em que o dispositivo da IoT coletou o dado e δ_{ib} é o tipo de dado do dispositivo da IoT que forneceu o dado coletado. A definição de cada tipo de dado $\delta_{e_{ik}} \in D_{e_i}$, $k > 0$, é dada pelos dispositivos da IoT conectados a e_i.

Cada nó de borda e_i possui um conjunto de μ nós virtuais $V_{e_i} = \{v_{ij} | i \in 0..n \wedge j \in 0..\mu \wedge e_i \in E\}$. O nó virtual é definido como uma tupla $v_{ij} = (m_{v_{ij}}, s_{v_{ij}}, \delta_{ij})$, na qual $m_{e_{ij}}$ é a memória alocada para o nó virtual e $s_{e_{ij}}$ é o espaço de armazenamento no disco do nó de borda e_i que é destinado ao nó virtual v_{ij}, e $\delta_{ij} \in D_{e_i}$ é o tipo de dado de v_{ij}. Durante qualquer momento, a quantidade de memória e espaço de armazenamento designada para os nós virtuais é limitada pela quantidade de memória e espaço de armazenamento do nó de borda, conforme descrito pela expressão

$$\forall e_i \in E, \sum_{j=0}^{\mu} m_{v_{ij}} < M_{e_i} \wedge \sum_{j=0}^{\mu} s_{v_{ij}} < S_{e_i}. \tag{5.1}$$

Uma requisição r_k chega em um nó de borda de um dispositivo da IoT e é definida como uma tupla $r_k = (I_{r_k}, m_{r_k}, s_{r_k}, \delta_{r_k}, p_{r_k}, f_{r_k}, t_{0k})$, para $k > 0$, onde I_{r_k} é o número de instruções a serem executadas, m_{r_k} é a quantidade de memória, s_{r_k} é o espaço de armazenamento, δ_{r_k} é o tipo de dado da requisição, p_{r_k} é o valor da prioridade da requisição, f_{r_k} é o requisito de frescor de dado da requisição e t_{0k} é o momento em que a requisição foi gerada. Entre as várias definições de frescor de dado, neste trabalho nós calculamos o requisito de frescor de um dado como o tempo decorrido desde a sua aquisição no nível de coisas até o momento em que o dado é requisitado para processamento no nó de borda (BOUZEGHOUB (2004)).

Para acomodar a situação em que o provedor de infraestrutura deseja maximizar

o valor pago pelo usuário e fornecer um serviço diferenciado de alta qualidade para cada usuário final da aplicação, o nível de prioridade de requisição p_{r_k} é diretamente derivada do valor financeiro pago pelo usuário final que gerou r_k.

O valor de p_{r_k} é usado no modelo de nó virtual. Seu objetivo é garantir que os usuários finais que desejam pagar por uma maior qualidade de serviço possam ser atendidos adequadamente. A prioridade da requisição é definida como $p_{r_k} = \psi_{r_k}/\Psi$, para $0 \leq \psi_{r_k} \leq \Psi$, onde ψ_{r_k} é o valor financeiro pago pelo usuário final que gerou r_k e Ψ é o valor financeiro máximo estabelecido pelo provedor de infraestrutura.

5.3 Infraestrutura na Camada de Borda

Nós assumimos que a camada de borda pertence e é administrada por um provedor de infraestrutura. O provedor de infraestrutura define os limites físicos e administrativos (lógicos) das redes sem fio conectadas aos nós de borda. Além disso, ele também configura as conexões existentes entre os dispositivos da IoT e os nós de borda, além das conexões existentes, e a forma de comunicação , entre os nós da camada de borda. Nós assumimos que um dispositivo da IoT se conecta ao nó de borda mais próximo a ele geograficamente.

O gerenciamento da infraestrutura é realizado pelo provedor de infraestrutura, o qual investe recursos financeiros e opera os dispositivos que abrangem o nível de borda para dar suporte aos serviços requisitados pelas aplicações. A infraestrutura consiste em todos os recursos de processamento, armazenamento e comunicação dos dispositivos que compõem a camada de borda e são necessários para executar os serviços de alocação de recursos requeridos pelas aplicações. A alocação de recursos proposta neste trabalho abrange o recebimento do dado coletado na camada das coisas, o processamento do dado na camada da borda e a entrega do dado, respeitando os requisitos da requisição, para a aplicação.

O provedor de infraestrutura também é responsável por definir os tipos de dados suportados por cada nó de borda, de acordo com os dispositivos da IoT conectados ao respectivo nó de borda. O provedor de infraestrutura configura os níveis de prioridade disponíveis, bem como a maneira como os níveis de prioridade são estabelecidos. Em nosso modelo de virtualização, o provedor de infraestrutura define o nível de prioridade baseado no valor financeiro pago pelo usuário.

O Algoritmo 4 mostra o pseudo-código do algoritmo desenvolvido para organizar os nós na camada de borda chamado Algoritmo de Criação de Infraestrutura de Camada de Borda (ACICB) proposto como parte de nosso trabalho. O ACICB escolhe o nó pai de cada nó com base no problema de caminho mais curto entre todos os pares de nós. Esse algoritmo possui as variáveis n, $nmaster$, $coords$, t, C, M, S e W como entrada e retorna as variáveis $parent$ e $master$ como saída.

Algorithm 4: Algoritmo de Criação de Infraestrutura de Camada de Borda

Input : $n, nmaster, coords, t, C, M, S, W$
Output : $master, parent$

if $t = cap$ **then**
  $master \leftarrow get\_masters\_cap(nmaster, C, M, S, W)$
foreach $i \in \{1..n\}$ **do**
  foreach $j \in \{1..n\}$ **do**
    if $i = j$ **then**
      $A_{ij} \leftarrow 0$
    else if $d(c_i, c_j) > 0$ **then**
      $A_{ij} \leftarrow d(c_i, c_j)$
    else
      $A_{ij} \leftarrow \infty$
$minmax \leftarrow \infty$;
foreach $k \in \{1..n\}$ **do**
  foreach $i \in \{1..n\}$ **do**
    $H_i \leftarrow 0$;
    foreach $j \in \{1..n\}$ **do**
      if $A_{ik} + A_{kj} < A_{ij}$ **then**
        $A_{ij} \leftarrow A_{ik} + A_{kj}$;
      if $A_{ij} > H_i$ **then**
        $H_i \leftarrow A_{ij}$
    if $t = dist$ **then**
      $master \leftarrow get\_masters\_dist(nmaster, H)$;
    $root_i \leftarrow get\_root\_min(A, i, master)$;
    if $master_i$ **then**
      $parent_i \leftarrow i$
    else
      $parent_i \leftarrow get\_neighbor\_min(A, i, root_i)$;

A variável *nmaster* é o número de nós de borda mestre. A variável $coords_i = (x_i, y_i)$ é a coordenada geográfica do nó de borda e_i, para valores positivos de x_i e y_i, e $coords = \{coords_i, \forall i\}$. A variável $t = \{cap, dist\}$ é a metodologia de escolha dos nós mestres (capacidade ou distância). $C = \{C_i, \forall i > 0\}$ é o conjunto de taxas de processamento, $M = \{M_i, \forall i > 0\}$ é o conjunto das tamanhos de memória, $S = \{S_i, \forall i > 0\}$ é o conjunto de espaço de armazenamento permanentes, para todos os nós de borda. A variável W_i corresponde a soma da largura de banda de todos os links do nó de borda e_i, e $W = \{W_i, \forall i > 0\}$ é o conjunto da soma da largura de banda de todos os links, para todo nó de borda e_i. Cada nó de borda e_i possui um índice de capacidade P_i, que corresponde a soma ponderada de C_i, M_i, S_i e W_i, e é definido como

$$P_i = C_i\alpha_C + M_i\alpha_M + S_i\alpha_S + W_i\alpha_W. \tag{5.2}$$

Na expressão 5.2, α_C, α_M, α_S e α_W são coeficientes positivos de taxa de processamento, memória, armazenamento e largura de banda, respectivamente. O provedor de infraestrutura determina o valor de cada coeficiente de acordo com a importância dada a cada recurso.

Quando a metodologia para escolher nós mestres é $t = cap$, o ACICB chama a função "get_masters_cap"que retorna os nós mestres com o maior valor do índice P_i. Se a metodologia for $t = dist$, os *nmaster* nós que minimizam a distância de todos os outros nós são escolhidos como mestres. A variável $parent_i$ indica qual vizinho do nó de borda e_i é o nó pai de e_i, ou seja, a variável $parent_i$ determina a hierarquia dos nós de borda e $parent = \{parent_i, \forall i\}$. A variável $master_i$ possui um tipo booleano, que indica se o nó de borda e_i é um nó mestre e $master = \{master_i, \forall i\}$.

Após a execução do algoritmo, se $master_i$ é igual a verdadeiro, então o nó e_i é um nó mestre, ou seja, $\tau_i = MASTER$. A função d é a função de distância euclidiana entre dois nós de borda. As linhas 1-2 do Algoritmo 4 verificam se a escolha dos nós mestres deve ser realiza por capacidade e, em um caso afirmativo, a função "get_masters_cap"é chamada. A função "get_masters_cap"calcula o valor de P_i para cada nó de borda e_i e escolhe os *nmaster* nós de borda com o maior valor de P_i.

As linhas 3-10 do algoritmo correspondem à fase de inicialização do algoritmo, onde a matriz A, usada para calcular o caminho mais curto entre todos os pares de nós, é inicializada. As linhas 11-18 calculam o caminho mais curto entre todos os pares de nós e armazena a maior distância do nó de borda e_i em H_i. As linhas 19 a 20 verificam se o nó mestre deve ser escolhido pelo critério de distância e, em caso afirmativo, chamam "get_masters_dist". A função "get_masters_dist"escolhe os *nmaster* nós de borda com o menor valor de H_i. As linhas 21-25 calculam $parent_i$

para todos os nós de borda e_i. A função "get_root_min (A, i, $master$)"retorna o nó mestre mais próximo do nó de borda e_i. A função "get_neighbor_min (A, i, $root_i$)"retorna o vizinho do nó de borda e_i, que possui o caminho mais curto para o nó mestre $root_i$.

No caso de uma falha no nó de borda e_i, se $e_i = parent_j$ para algum nó de borda e_j, então e_j chama "get_neighbor_min (A, i, $root\_i$)"para escolher o novo nó pai. Se e_i for um nó de borda mestre, para todos os nós de borda e_j tal que $e_i = root_j$, então e_j chama "get_root_min $(A, i, root_i)$"para escolher o novo nó de borda mestre. No caso de um novo nó de borda e_i se anexar à rede, e_i chama "get_-neighbor_min $(A, i, root_i)$"para escolher seu nó pai. O procedimento em casos de falha é o mesmo para nós de borda desconectados (desligados, por exemplo) da rede e o procedimento para novo nó é o mesmo para nós de borda reconectados à rede. O ACICB é executado novamente quando ocorre uma falha em um nó de borda mestre e nenhum outro nó de borda mestre permanece ativo na rede.

5.4 Algoritmo de Alocação de Recursos

O algoritmo de alocação de recursos proposto simula um modelo de agentes orientados a mercado inspirado no trabalho de MULLEN e WELLMAN (1996). O modelo descrito pelos autores consiste da existência de agentes econômicos que trocam um bem financeiro com o objetivo de resolver um problema de utilidade. Em nosso algoritmo, cada nó de borda assume o papel de agente econômico e o bem a ser trocado corresponde a uma alocação de uma requisição de aplicação para um nó virtual. Cada alocação de requisição para um nó virtual possui um valor de utilidade e o problema resolvido pelo nó de borda consiste em escolher o nó virtual que maximiza o valor da função de utilidade para alguma requisição. Na Seção 5.4.1, nós detalhamos o modelo de nó virtual baseado em mercado de agentes econômicos.

Nosso Algoritmo Colaborativo de Alocação de Recursos (ACAR) determina onde cada requisição de aplicação enviada ao sistema da CoT vai ser atendida. O ACAR faz a alocação de recursos empregando dois mecanismos: um processo de encaminhamento de requisição e um mecanismo de mercado nós virtuais.

Quando uma requisição r_k chega em um nó de borda e_i, o ACAR determina localmente, qual é o nó virtual mais vantajoso para atender a requisição. Para fazer isso, o ACAR avalia todos os nós virtuais já instanciados e avalia a possibilidade de instanciar um novo nó virtual. Para tomar a decisão de alocar a requisição para um determinado nó virtual, o nó de borda calcula o valor de utilidade do nó virtual. Esse valor de utilidade é baseado no nível de prioridade da requisição e em um valor de estimativa do custo total gerado pelo nó virtual escolhido para atender r_k. Se nenhum nó virtual pertencente a e_i for capaz de satisfazer os requisitos da requisição

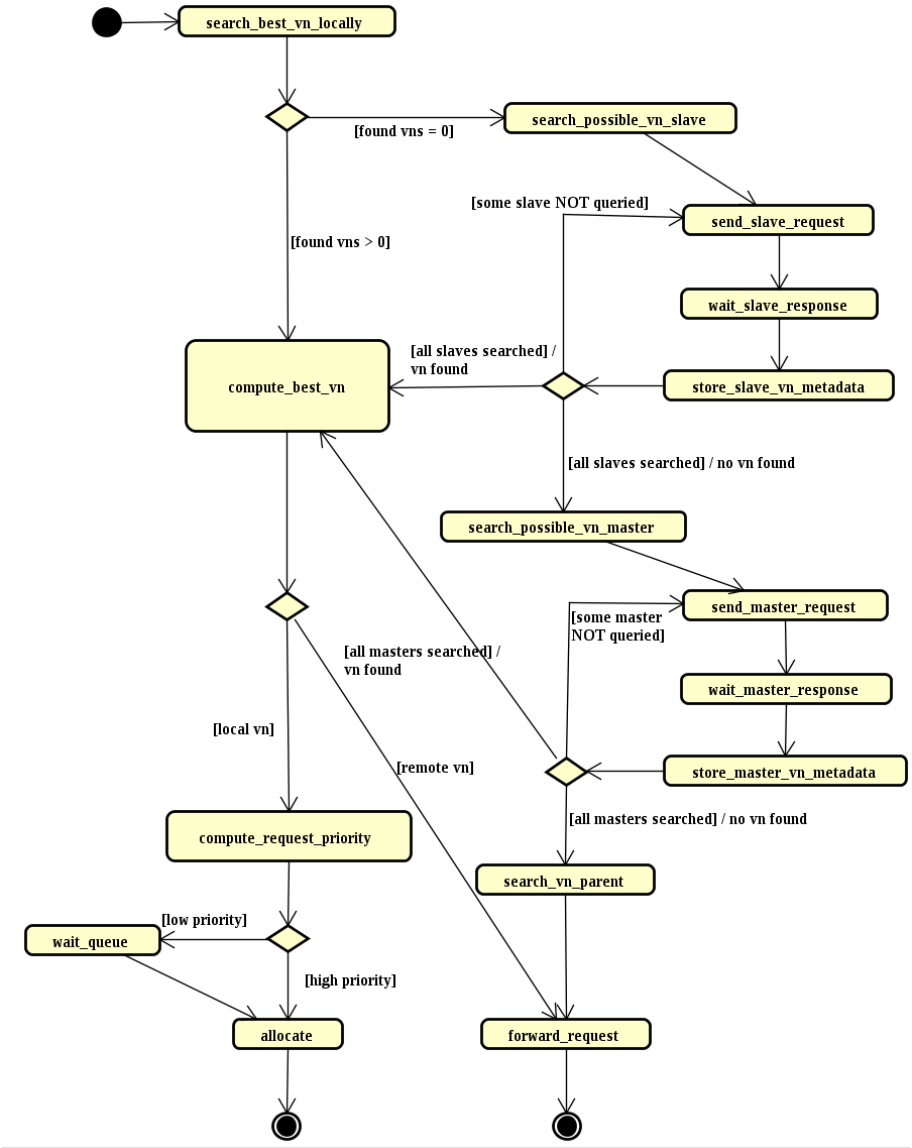


Figura 5.2: Diagrama de Atividades do Algoritmo Colaborativo de Alocação de Recursos.

r_k, a requisição é encaminhada para um nó escravo ou para o nó mestre.

A Figura 5.2 apresenta o diagrama de atividades do algoritmo de alocação de recurso proposto. Quando uma nova requisição r_k chega em um nó de borda e_i, a sequência de atividades nesta figura é executada no sistema. Inicialmente, uma requisição chega em um determinado nó de borda (atividade inicial) e a atividade "search_best_vn_locally"é executada. Esta atividade procura em V_{e_i}, o conjunto de nós virtuais instanciados de e_i, em busca de um nó virtual que atenda os requisitos da requisição r_k. Se nenhum nó virtual for encontrado, a atividade "search_possible_-vn_slave"é executada; caso contrário, se algum nó virtual for encontrado, "compute_-best_vn"é chamado (este método será explicado em detalhes abaixo).

A atividade "search_possible_vn_slave"procura por nós virtuais em todos os nós

escravos conectados ao nó de borda atual, se existirem. Essa busca nos nós escravos é feita por meio de três atividades: "send_slave_request", "wait_slave_response"e "store_slave_vn_metadata". Ao executar essas atividades, o nó de borda envia a mensagem de consulta, aguarda uma resposta e armazena os parâmetros significativos do nó virtual (metadados do nó virtual) de seus nós escravos. Essas três atividades são repetidas até que todos os nós escravos sejam consultados. Quando todos os nós escravos foram consultados, há dois resultados possíveis: (i) se nenhum nó virtual foi encontrado, porque nenhum dos escravos pode atender a requisição, a atividade "search_possible_vn_master"é executada, (ii) caso contrário, "compute_best_vn"é chamado.

A atividade denominada "search_possible_vn_master"é semelhante a "search_possible_vn_slave", mas ela pesquisa apenas os nós de borda que são possivelmente nós mestres. Por meio desta atividade, um nó mestre pode realizar colaboração com outros nós mestres, de maneira que a requisição é encaminhada para a subárvore do outro nó de borda mestre. A atividade "search_possible_vn_master"também é seguida por três atividades: "send_master_request", "wait_master_response"e "store_master_vn_metadata". Se todos os nós mestres foram consultados e pelo menos um nó virtual foi encontrado, "compute_best_vn"é chamado; caso contrário, a atividade "search_vn_parent"é executada.

A atividade "compute_best_vn"possui duas funções: acessar o conjunto local de nós virtuais para selecionar o melhor deles a ser alocado para a requisição recebida; ou preparar o encaminhamento da requisição para um nó remoto (mestre ou escravo). No primeiro caso, a seleção do melhor nó virtual é realizada por meio da avaliação do valor de utilidade dos nós virtuais. O valor de utilidade é calculado por meio da abordagem baseada em mercado proposta e descrita na Seção 5.4.1.

Se algum nó virtual local for selecionado como o melhor, a atividade "compute_request_priority"é executada, caso contrário, a atividade "forward_request"é chamada. A atividade "compute_request_priority"calcula uma probabilidade do nó virtual atender à requisição com base na prioridade. Se a requisição possuir baixa prioridade, ela aguarda na fila de requisições (atividade "wait_queue"). Caso contrário, o nó de borda executa a atividade "allocate"para alocar o nó virtual escolhido para a requisição. A atividade "search_vn_parent"verifica o nó pai do nó de borda atual, o qual pode ser outro nó de borda ou a nuvem. Depois disso, a atividade "forward_request"é chamada para enviar a requisição para um nó de borda ou para a nuvem.

O Algoritmo 5 mostra o pseudocódigo do ACAR. Quando uma requisição r_k chega ao nó de borda e_i, a função "search_best_vn_locally"é chamada (linha 5) para procurar localmente uma lista dos possíveis nós virtuais para atender a requisição. Na lista de nós virtuais, o nó de borda calcula aquele cujo valor de utilidade fornecido

Algorithm 5: Algoritmo Colaborativo de Alocação de Recursos

```
1  while true do
2      event ← receiveEvent();
3      if event.type = REQUEST_ARRIVED then
4          r_k ← (Request)event.message;
5          vnList ← search_best_vn_locally (r_k);
6          v_ij ←compute_best_vn (vnList);
7          if v_ij is found then
8              if ρ < U(v_ij, r_k) then
9                  allocate (v_ij, r_k);
10             else
11                 push (Q_{e_i}), r_k);
12         else
13             bestSlave = search_possible_vn_slave (r_k);
14             if bestSlave is found then
15                 v_ij ← compute_best_vn (bestSlave.vnList);
16                 foward_request (bestSlave, (v_ij, r_k));
17             else
18                 if τ_{e_i} = MASTER then
19                     bestMaster = search_possible_vn_master (r_k);
20                     if bestMaster is found then
21                         v_ij ← compute_best_vn (bestMaster.vnList);
22                         forward_request (bestMaster, (v_ij, r_k));
23                     else
24                         forward_request (cloud, r_k);
25                 if τ_{e_i} = SLAVE then
26                     parent = search_possible_vn_parent (r_k);
27                     foward_request (parent, r_k);
28     if event.type = VN_FINISHED then
29         v_ij ← (VN) event.message;
30         update_resources (v_ij);
31     if event.type = TIMEOUT then
32         no_allocation = True;
33         while no_allocation and |Q_{e_i}| > 0 do
34             foreach r_k ∈ Q_{e_i} do
35                 v_ij ← search_best_vn_locally(r_k);
36                 if v_ij is found and ρ > U(v_ij, r_k) then
37                     allocate (v_j, r_k);
38                     no_allocation ← False;
```

pela função $U(v_{ij}r_k)$ é o máximo. Portanto, se o valor do utilidade do nó virtual for maior que a probabilidade ρ, a requisição r_k será alocada para v_{ij}. Caso contrário, r_k é armazenado na fila Q_{e_i}.

Os nós escravos são consultados por meio da função "search_possible_vn_slave"e a requisição é encaminhada para algum nó escravo (linha 12) se nenhum nó virtual for encontrado localmente. Quando a requisição chega em um nó mestre (linha 16), ela pode fazer colaboração com outros nós mestres (linhas 17 a 19) ou ser encaminhada para a nuvem. No primeiro caso, a requisição é encaminhada para outro nó mestre (linha 17) que tentará aloca-lá para o melhor nó virtual de sua subárvore.

Finalmente, se uma requisição chega em um nó escravo, o nó de borda encaminha a requisição para seu pai (linhas 22-24), que possui o nível mais alto na hierarquia. Quando um nó virtual termina de atender sua requisição, ele interrompe o nó de borda (linha 25) que atualiza seus recursos e libera recursos para atender a novas requisições (linha 27). Um valor de tempo limite é configurado e periodicamente executa o processamento da fila de requisições Q_{e_i} do nó de borda. O nó de borda executa o loop nas linhas 30-35 e permanece nele que até alocar alguma requisição para um nó virtual ou a fila Q_{e_i} estiver vazia. Dentro do loop, o nó de borda é interrompido sempre que uma requisição chega ou um nó virtual é finalizado.

5.4.1 Modelo de Mercado de Nós Virtuais

Nós adotamos uma abordagem inspirada em modelo de agentes econômicos para determinar o valor de alocações de requisições para nós virtuais na camada de borda. Em nosso modelo os nós de borda são como agentes econômicos e o bem negociado são alocações de requisições, as quais possuem um valor de utilidade. Na chegada de uma requisição, o nó de borda calcula o valor de utilidade localmente para todos os nós virtuais já instanciados anteriormente. O nó de borda também procura em seus vizinhos quando ele não possui recursos disponíveis para atender a requisição. Por meio da avaliação da função de utilidade, o nó de borda pode optar encaminhar a requisição para um vizinho.

A função de utilidade é composta pelo preço de requisição e o custo estimado da requisição ser alocada pelo nó virtual. O preço é uma função do valor financeiro pago pelo usuário para ter seu serviço atendido. O custo de alocação, calculado pelo nó de borda, corresponde a uma estimativa do custo de alocar a requisição para um determinado nó virtual. Este custo leva em consideração métricas de desempenho relacionadas ao mecanismo de virtualização e a comunicação de rede. Por meio do modelo de mercado de nós virtuais, o nó de borda é capaz de distribuir de maneira homogênea os recursos de rede disponíveis na camada de borda, além de respeitar os níveis de prioridades das aplicações.

Quando uma requisição r_k chega em um nó de borda e_i, o nó de borda decide se r_k será atendida localmente ou em outro nó de borda. Essa decisão inicial é tomada na função "search_best_vn_locally"e é baseada na capacidade computacional do nó e no tipo de dado da requisição e no conjunto de tipos de dados de e_i. A capacidade computacional refere-se à taxa do processador, a quantidade de memória disponível e a quantidade de espaço de armazenamento disponível. O tipo de dado da requisição também necessita estar presente no conjunto de tipos de dados D_{e_i} do nó de borda e_i. Dessa forma, as três restrições a seguir devem ser atendidas

$$m_{r_i} < M_{e_i} - \sum_{j=0}^{\mu} m_{v_{ij}} \tag{5.3}$$

$$s_{r_k} < S_{e_i} - \sum_{j=0}^{\mu} s_{v_{ij}} \tag{5.4}$$

$$\delta_{r_k} \in D_{e_i} \tag{5.5}$$

As expressões 5.3 e 5.4 garantem que os recursos de memória e armazenamento disponíveis de e_i são suficientes para atender aos requisitos de r_k, e a expressão 5.5 garante que o nó de borda tenha o tipo de dado da requisição. Se pelo menos uma dessas restrições não for válida, o nó e_i encaminha a requisição r_k para outro nó de borda. Se as restrições forem válidas, o nó de borda calcula localmente o melhor nó virtual para atender r_k. Para obter o valor de utilidade de uma alocação de nó virtual para uma requisição, o nó de borda e_i calcula o valor da função U para o nó virtual v_{ij} como na expressão a seguir

$$U(v_{ij}, r_k) = \frac{p_{r_k}}{C(v_{ij}, r_k)}. \tag{5.6}$$

Na expressão 5.6, p_{r_k} é a prioridade da requisição e $C(v_{ij}, r_k)$ é um custo total para v_{ij} atender a requisição r_k. O nó virtual com o maior valor de utilidade é escolhido para atender r_k, isto é, $max_{j \in 0..\mu}(U(v_{ij}, r_k))$. A função $C(v_{ij}, r_k)$ é calculada como a soma normalizada dos cinco custos na expressão

$$C(v_{ij}, r_k) = [C_r(v_{ij}) + C_i(v_{ij}) + C_p(v_{ij}) + C_u(v_(ij)r_k) + C_q(v_{ij}, r_k)]/C_{M_i}. \tag{5.7}$$

As funções de custo que compõem $C(v_{ij})$ são: custo de reconfiguração $C_r(v_{ij})$, custo de instanciação $C_i(v_{ij})$, custo de processamento $C_p(v_{ij})$, custo de atualização $C_u(v_{ij}, r_k)$ e custo de fila $C_q(v_{ij}, r_k)$. A soma dos custos é normalizada por C_{M_i}, o valor máximo de $C(v_{ij}, r_k)$ para todos os nós virtuais já instanciados pelo nó de borda e_i. Quando o nó de borda e_i recebe r_k, e_i precisa escolher o nó virtual para atender

r_k. Se o nó virtual escolhido já estiver instanciado, mas os recursos solicitados forem maiores que os recursos disponíveis no nó virtual, será necessário reconfigurar a quantidade de memória $m_{v_{ij}}$ e o espaço de armazenamento $s_{v_{ij}}$. Reconfigurar um nó virtual pode ser mais vantajoso do que instanciar um novo nó virtual. O custo de reconfiguração é então definido como na expressão a seguir

$$C_r(v_{ij}) = \begin{cases} 0, & \text{se } v_{ij} \text{ não está instanciado} \\ 0, & \text{se } v_{ij} \text{ está instanciado} \wedge \text{reconf. não necessário} \\ \Delta_r(v_{ij}) & \text{se } v_{ij} \text{ está instanciado} \wedge \text{reconf. necessário} \end{cases} \tag{5.8}$$

Na expressão 5.8, $\Delta_r(v_{ij})$ é o tempo de reconfiguração médio de um nó virtual de e_i. A computação do valor $\Delta_r(v_{ij})$ pode ser feita com pouco custo, porque o nó de borda pode manter uma variável para armazenar a soma do tempo das reconfigurações e outra para armazenar a contagem de reconfigurações. Assim, o nó de borda pode calcular o tempo médio de reconfiguração com operações aritméticas de custo computacional baixo. $C_i(v_{ij})$ é o custo de instanciação de um novo nó virtual. Quando uma requisição chega no nó de borda, ele calcula o custo de possivelmente instanciar um nó virtual. O custo da instanciação é então definido como na expressão a seguir

$$C_i(v_{ij}) = \begin{cases} 0, & \text{se não instancia novo vn} \\ \Delta_i(v_{ij}) & \text{se instancia novo vn} \end{cases} \tag{5.9}$$

Na expressão 5.9, $\Delta_i(v_{ij})$ é o tempo médio de instanciação de um nó virtual no nó de borda e_i. Semelhante a $\Delta_r(v_{ij})$, o nó de borda pode calcular $\Delta_i(v_{ij})$ com poucas operações aritméticas, mantendo uma variável para armazenar a soma dos tempos de instanciação e outra para armazenar o número de nós virtuais instanciados. $C_p(v_{ij})$ é uma estimativa do custo de processamento da requisição no nó virtual v_{ij}. Essa variável corresponde à estimativa do tempo necessário para o nó de borda executar todas as instruções de requisição I_{r_k} e é definida como na expressão

$$C_p(v_{ij}, r_k) = \frac{I_{r_k}}{C_{e_i}} \tag{5.10}$$

Na expressão 5.10, C_{e_i} é a taxa de processamento do nó de borda e_i. O tempo de execução de um nó virtual pode ser calculado por meio de amostragem. No entanto, usando o tamanho da requisição em termos de número de instruções, é possível obter estimativas mais efetivas para $C_p(v_{ij}, r_k)$.

$C_u(v_{ij})$ é o custo estimado para obter o dado requirido pela requisição a partir da camada das coisas ou da cache local. Antes de alocar uma requisição a um nó

virtual, o nó de borda e_i verifica se algum dado da cache local possui o mesmo tipo de dado da requisição. Além disso, e_i verifica se o tempo de aquisição do dado corresponde ao requisito de frescor do dado da requisição. Se a verificação for bem-sucedida, $C_u(v_{ij}) = 0$, porque o dado será obtido da cache local rapidamente. Se a verificação falhar, o tempo médio de atraso para coletar o dado do dispositivo da IoT é calculado. Assim, $C_u(v_{ij})$ é definido como

$$C_u(v_{ij}, r_k) = \begin{cases} 0, & \text{se } \delta_{r_k} \in \{\delta | \forall (t_{ib}, \delta_{ib}) \in B_{e_i} \wedge f_{r_k} < F(t_{ib})\} \\ \Delta_u(v_{ij}) & \text{caso contrário} \end{cases} \tag{5.11}$$

Na expressão 5.11, $F(t_{ib})$ é uma função para calcular o frescor do dado em cache b, Δ_u é uma função que computa o tempo médio de atraso para coletar o dado do dispositivo na camada das coisas.

$C_q(v_{ij}, r_k)$ é o tempo médio da fila que r_k espera para chegar em e_i. Este custo é calculado como na expressão a seguir

$$C_q(v_{ij}, r_k) = \frac{1}{m} \sum_{k=0}^{m} T_{ik}(t_{0r_k}) \tag{5.12}$$

Na expressão 5.12, T_{ik} é uma função que calcula a diferença de tempo entre o momento quando o nó de borda e_i recebe r_k e t_{0r_k}, o qual é o momento quando a aplicação gerou a requisição.

Capítulo 6

Avaliação do Algoritmo de Alocação de Recursos

Este capítulo apresenta a avaliação experimental da solução de alocação de recursos descrita no Capítulo 5. Na Seção 6.1, nós descrevemos a configuração experimental, na Seção 6.2 nós apresentamos a metodologia de avaliação para os experimentos e na Seção 6.3 nós apresentamos os resultados obtidos.

6.1 Configuração Experimental

A fim de avaliar o desempenho dos sistemas de CoT de duas e três camadas, nós realizamos simulações utilizando o simulador de eventos discretos (DES - Discrete Event Simulator) YAFS, de LERA *et al.* (2019). Nós usamos esse simulador porque ele é especialmente projetado para cenários da CoT, além de ser robusto e estável. Nós usamos o núcleo do simulador para obter a simulação de atrasos, filas e recursos compartilhados da rede, mas tivemos que estender o simulador para suportar todos os nossos cenários e características de infraestrutura.

Quatro cenários arquiteturais diferentes foram avaliados, conforme descrito abaixo:

- Nuvem: corresponde ao cenário de duas camadas. Não possui a camada de borda e os dispositivos da IoT são conectados diretamente a um datacenter na nuvem.
- Vertical: corresponde ao sistema de CoT de três camadas, em que todos os nós de borda possuem links de comunicação com a camada de nuvem, porém não há links entre os nós de borda.
- Horizontal: corresponde a uma versão estendida do modelo Vertical, na qual os nós de borda possuem conexões entre si, formando uma topologia em linha.

Devido a existência das conexões entre os nós de borda, eles podem dividir a carga de trabalho entre si para atender as requisições das aplicações de maneira distribuída.

- Hierárquico: corresponde a nossa proposta de arquitetura, consistindo de um sistema da CoT de três camadas, em que os nós de borda são organizados hierarquicamente por meio do Algoritmo 4. Nós configuramos a camada de borda com um nó mestre, pois o objetivo é entender o impacto do algoritmo em reduzir o tempo de resposta das requisições, o que seria mais complexo em um cenário com vários nós mestres. Devido a isso, conforme o Algoritmo 4, o nó mestre corresponde ao nó mais equidistante de todos os outros nós de borda. A metodologia de escolha dos nós mestre foi configurada como $t = dist$.

Como premissa, um nó da nuvem sempre possui a capacidade necessária para atender a requisição, em termos de memória e capacidade de processamento. Por outro lado, o tempo de processamento e os recursos de memória dos nós da camada de borda e da camada das coisas são limitados. A seguir, descrevemos como nossos cenários experimentais foram configurados e a Tabela 6.1 apresenta um resumo dos parâmetros adotados.

Coordenadas - As coordenadas geográficas (latitude e longitude) dos nós das três camadas foram escolhidas aleatoriamente em uma área quadrada limitada em um plano cartesiano com origem no ponto (0,0) e valores máximos limitados pelas variáveis x_{max} e y_{max}.

Topologia - No cenário Nuvem, todos os dispositivos da IoT foram conectados à nuvem. Nos cenários de três camadas, cada dispositivo da IoT foi conectado a um nó de borda e cada conexão entre os nós da camada de coisas e os nós da camada de borda foi determinada pelo bem conhecido algoritmo de diagrama de Voronoi, conforme proposto pelos autores XU *et al.* (2017).

Dadas as coordenadas geográficas dos nós de borda, foi gerado um diagrama de Voronoi, o qual retorna a área de cobertura geográfica de cada nó de borda. Essa área corresponde à localização geográfica onde o nó de borda mantém suas conexões com os dispositivos da IoT conectados a ele. Um dispositivo da IoT é conectado ao nó de borda se sua coordenada geográfica estiver na área geográfica controlada pelo nó de borda. Por meio do diagrama de Voronoi, o dispositivo da IoT se conecta ao nó de borda geograficamente mais próximo a ele.

Na camada de borda, as conexões topológicas entre os nós de borda foram estabelecidas de maneira diferente, de acordo com cada cenário. O cenário Vertical não possui conexões entre nós de borda. No cenário Horizontal, os links de comunicação entre os nós de borda foram gerados para criar uma topologia de linha na camada de borda. No cenário Hierarquia, os links de comunicação foram criados

aleatoriamente, de maneira a criar um grafo conectado da topologia.

Além disso, os links de comunicação para a nuvem foram determinados da seguinte maneira. Nos cenários Vertical e Horizontal, todos os nós de borda foram conectados à nuvem. No cenário Hierárquico, apenas os nós de borda mestre, os quais foram determinados pelo Algoritmo 4, foram configurados com link de comunicação com a nuvem.

Recursos - Os recursos de um nó abrangem a largura de banda, a taxa de CPU, a memória e o espaço de armazenamento. Nós adotamos a mesma configuração de largura de banda descrita pelos autores YOUSEFPOUR *et al.* (2018) para nossos experimentos. É pressuposto que um nó da camada das coisas se comunique com a camada de borda ou com a camada de nuvem (dependendo do cenário) usando o protocolo IEEE Wifi 802.11 com largura de banda de 54 Mbps. Os links entre os nós de borda são considerados links Ethernet e entre o nós da camada de borda e a nuvem os links são Gigabit Ethernet (a Tabela 6.1 apresenta os valores específicos).

Os dispositivos da IoT são considerados heterogêneos. Atualmente, existe uma grande variedade de hardware de sensor disponível no mercado e nos experimentos, nós consideramos que o dispositivo da IoT possuem arquitetura Arduino (NAYYAR e PURI (2016), com nós dotados de uma taxa de CPU de 16 MHz e 32KB de memória. Os nós de borda foram configurados com taxa de CPU de 1 GHz e o nó de nuvem possui taxa de CPU de 3 GHz. Com relação à memória, nós configuramos dois tipos de nós de borda heterogêneos, nos quais o primeiro tipo possui metade da capacidade de memória em comparação com o segundo tipo (a Tabela 6.1 mostra os valores). Os recursos de memória da nuvem foram configurados para serem sempre suficientes para atender as requisições em termos de memória e espaço de armazenamento.

Tipos de dado - Em nossos cenários simulados, consideramos dez tipos diferentes de dados que são fornecidos pela infraestrutura da CoT e são solicitados pelas aplicações. Os dez tipos de dados são distribuídos uniformemente para todos os dispositivos da camada das coisas. Cada dispositivo da IoT possui apenas um tipo de dado. O tipo de dado possui um tempo médio de atualização do dado (data update), o que caracteriza heterogeneidade para os dispositivos da IoT. Cada valor de atualização do dado representa o tempo necessário para o dispositivo da IoT obter e enviar o dado. Os valores do tempo de atualização do dado foram distribuídos aleatoriamente e uniformemente na faixa de [0.1, 1.5] segundos.

Requisições - Nos cenários de três camadas (Vertical, Horizontal e Hierarquia), todos os nós da camada de borda são pontos de entrada para as requisições das aplicações. O número total de requisições geradas (Tabela 6.1) é distribuído uniformemente entre todos os nós de borda. O tempo de chegada das requisições é distribuído exponencialmente ao longo de todo o tempo da simulação. Os tipos de

dados foram distribuídos uniformemente entre as requisições e usamos o tamanho de requisições como descrito em YOUSEFPOUR *et al.* (2018), cujos valores estão na Tabela 6.1.

Com relação as características das requisições, consideramos um cenário típico borda-nuvem, onde diferentes usuários enviam requisições pertencentes a domínios de aplicações heterogêneos. Ou seja, as aplicações são heterogêneas e demandam diferentes tipos de dados de sensoriamento e possuem diferentes requisitos de QoS. O comportamento do sistema é agnóstico ao domínio da aplicação. O valor financeiro ψ_{r_k} foi escolhido uniformemente a partir do conjunto $\{10, 25, 50, 75, 100\}$. A variável ψ_{r_k} e $\Psi = 100$ foram usadas para calcular o valor de prioridade da requisição p_{r_k}.

Parâmetro	**Valor**
Área máxima (x_{max}, y_{max})	10 x 10(km)
Número de rodadas de simulação	40 rodadas
Número total de requisições geradas	10000 requisições
Número de nós de borda (NEN)	20 nós
Taxa de chegada de requisições (RAR)	50 req./sec.
Frescor do dado (DFR)	100 ms
Taxa de CPU na nuvem	3 GHz
Taxa de CPU na camada de borda	1 Ghz
Memória do nó de borda	[1, 2] GB
Número de dispositivos da IoT por nó de borda	5 nodes
Atraso entre dispositivos da IoT e nuvem	35 ms
Atraso entre dispositivos da IoT e nós de borda	2 ms
Atraso entre nó de borda e nuvem	35 ms
Largura de banda entre nó de borda e nuvem	1 Gbps
Largura de banda entre nós de borda	100 Mbps
Largura de banda entre dispositivos da IoT e nuvem	54 Mbps
Largura de banda entre dispositivos da IoT e nós de borda	54 Mbps
Tamanho médio da carga de trabalho da requisição	80 KB
Número médio de instruções por requisição	$100x10^6$ inst.

Tabela 6.1: Parâmetros do ambiente de simulação.

6.2 Metodologia de Avaliação

Nós adotamos a metodologia Goal/Question/Metric (GQM) (BASILI (1992)) para ajudar no planejamento da avaliação realizada. A GQM é uma metodologia hierárquica (três níveis) com base na premissa de derivar métricas de um conjunto de perguntas e objetivos definidos. De acordo com a metodologia, primeiro nós definimos os objetivos a serem perseguidos.

Um objetivo representa o nível conceitual relacionado ao objetivo da medição (qual objeto e porque), a perspectiva (qual aspecto e quem) e as características de

ambiente (onde). Segundo, cada objetivo precisa ser refinado em várias perguntas. Uma pergunta representa o nível operacional usado para definir o objeto de medição (produto, processo, recurso). Finalmente, as métricas representam as informações no nível quantitativo que devem ser coletadas para responder a uma ou várias perguntas. De acordo com BASILI (1992), as métricas podem ser objetivas ou subjetivas. Métricas objetivas representam dados que dependem apenas do objeto que está sendo medido e não do ponto de vista de quem o está medindo. Métricas subjetivas representam dados que dependem do objeto que está sendo medido, bem como o ponto de vista de quem o está medindo.

Os objetivos de nosso trabalho são apresentados a seguir. O objetivo G1 é analisar o algoritmo de alocação de recursos proposto, com o propósito de avaliar seu desempenho e balanceamento de carga, com relação à redução do tempo de resposta da requisição e ao uso da largura de banda da rede. O objetivo G2 é analisar o algoritmo de alocação de recursos proposto, com o objetivo de avaliar seu desempenho, com relação à redução do número de requisições encaminhadas para a nuvem. O objetivo G3 é analisar o algoritmo de alocação de recursos proposto, com o objetivo de avaliar seu desempenho, com relação à redução do consumo de energia na camada das coisas. O objetivo G4 é analisar o algoritmo de alocação de recursos proposto, com o objetivo de avaliar seu desempenho, com relação à eficácia do mecanismo de prioridade. Desses quatro objetivos, nós derivamos 4 questões relevantes (Q1, Q2, Q3 e Q4) descritas na Tabela 6.2.

Questão	Descrição da Questão	**Objetivo**
Q1	O algoritmo proposto para o sistema da CoT de três camadas reduz o tempo de resposta para atendimento de requisições e ajuda a reduzir o tráfego de rede comparado com um sistema da CoT de duas camadas?	G1
Q2	O algoritmo proposto para o sistema da CoT de três camadas ajuda a reduzir o número de requisições encaminhadas para a nuvem?	G2
Q3	O algoritmo proposto para o sistema da CoT de três camadas ajuda a economizar energia dos dispositivos da IoT comparado com um sistema de duas camadas?	G3
Q4	O algoritmo proposto para um sistema CoT de três camadas atende requisições com maior nível de prioridade com uma maior precedência?	G4

Tabela 6.2: Questões e objetivos.

A Tabela 6.3 mostra as métricas definidas para responder as perguntas Q1, Q2, Q3 e Q4. Para responder a questão Q1, nós usamos as métricas Tempo de Resposta de Requisição (RRT - Request Response Time) e Tráfego de Mensagens de Rede (NMT - Network Messages Traffic). A métrica RRT indica o tempo decorrido desde

Métrica	Descrição da métrica	**Questão**
RRT	Request Response Time (RRT) - Diferença de tempo absoluta entre o momento em que a requisição foi gerada em um dispositivo da IoT até o momento quando o dispositivo recebe a respectiva resposta	Q1
NMT	Network Messages Traffic (NMT)- Número de mensagens nas filas dos canais de comunicação (camada de rede)	Q1
PRF	Percentage of Requests Forwarded to the Nuvem (PRF) - Número de requisições encaminhadas para a nuvem.	Q2
ECT	Energy Consumption at the Things Tier (ECT) - Soma da energia consumida por todos os dispositivos da IoT durante todo o tempo de simulação.	Q3
RRT-P	Request Response Time per Priority (RRT-P) – RRT por nível de priridade.	Q4

Tabela 6.3: Métricas utilizadas para responder as questões.

o instante em que uma requisição de aplicação é gerada em um dispositivo na camada de coisas até o instante em que o dispositivo recebe a resposta correspondente para aquela requisição. Esse intervalo inclui o tempo necessário para: enviar a requisição do dispositivo da IoT para a nuvem ou para o nó de borda (dependendo do cenário); executar o algoritmo de alocação de recursos; receber dados de sensoriamento da camada das coisas, se necessário; encaminhar a requisição para outros nós de borda, se necessário; e enviar o resultado da requisição de volta ao dispositivo de origem.

A métrica NMT denota o tráfego de mensagens nos canais de comunicação da camada de borda e/ou camada da nuvem durante o tempo de simulação. É desejável que esse valor seja o menor possível, pois significa que a rede possui uma condição de tráfego leve, evitando potencial congestionamento que pode resultar em altos atrasos. No simulador adotado, essa métrica corresponde ao número de mensagens transmitidas na rede e é calculado como a média da quantidade de mensagens que são mantidas nas filas dos nós de borda e nuvem devido as mensagens estarem: aguardando para serem transmitidas no link físico; ou aguardando o processamento pelo algoritmo de alocação de recursos na fila de requisições do nó de borda.

Para responder à pergunta Q2, nós consideramos a métrica Porcentagem de Requisições Encaminhadas para a Nuvem (PFR - Percentage of Requests Forwarded to the Nuvem). A métrica PFR é a proporção entre o número de requisições encaminhadas para a nuvem pelo número total de requisições geradas. Essa métrica indica o número de requisições não atendidas na camada de borda e que foram encaminhadas para a nuvem, devido à falta de recursos do nó de borda em atender a requisição.

Para responder à pergunta Q3, nós usamos a métrica Consumo de Energia na

Camada das Coisas (ECT - Energy Consumption at the Things Tier), a qual denota a soma de energia consumida por cada dispositivo da IoT para enviar e receber os dados de sensoriamento. Para calcular a energia dissipada, nós usamos o modelo de envio e recebimento de dados de dispositivos da IoT usados pelos autores LI *et al.* (2017). Nós usamos o modelo de transmissão de dados de rede para a camada das coisas descrito por LI *et al.* (2017), pois a maior parte da energia consumida pelo dispositivo da IoT advém da transmissão de dados da comunicação sem fio. Além disso, as tarefas de sensoriamento apresentam um consumo de energia aproximadamente constante e baixo, quando comparado com a tarefa de envio e recebimento de dados. O consumo de energia do dispositivo da IoT para enviar e receber dados no meio de comunicação sem fio é diferente; portanto, a energia consumida para enviar e receber um dado de tamanho l para outro dispositivo na distância d é dada por

$$E_{tx}(l, d) = E_{elec}l + \epsilon_{amp}ld^x \tag{6.1}$$

$$E_{rx}(l) = E_{elec}l \tag{6.2}$$

onde E_{elec} é a energia de dissipação de rádio e ϵ_{amp} é a dissipação de energia do amplificador de transmissão. Esses são parâmetros de hardware e x é uma constante que pode variar de 2 a 4; nos experimentos x = 2. A quantidade de energia consumida na camada coisas é dada por $E_{things} = E_{tx} + E_{rx}$. Para responder à pergunta Q4, nós avaliamos o Tempo de Resposta da Requisição por Prioridade (RRT-P - Request Response Time per Priority). Essa métrica é semelhante ao RRT, mas o RRT-P é calculado para cada valor de prioridade.

6.3 Resultados Experimentais

Esta seção apresenta os resultados dos experimentos realizados para responder as perguntas Q1, Q2, Q3 e Q4. Os fatores que afetam o desempenho dos sistemas da CoT são multivariáveis, portanto, nós avaliamos todas as métricas descritas na seção anterior observando a variação de três parâmetros diferentes: Taxa de Chegada de Requisição (RAR - Request Arriving Rate), Requisito de Frescor de Dado (DFR - Data Freshness Requirement) e Número de Nós de Borda (NEN - Number of Edge Nodes). Esses parâmetros são úteis para avaliar diferentes comportamentos e requisitos das aplicações e a escalabilidade do sistema.

A métrica RAR é definida como o número médio de requisições geradas por segundo e por nó de borda. Ela é usada para configurar o intervalo médio entre duas requisições consecutivas em um nó de borda, que é gerado a partir de uma distribuição exponencial. A métrica DFR é o requisito de frescor do dado reque-

rido pela aplicação e corresponde ao tempo máximo decorrido desde a aquisição do dado pelo dispositivo da IoT tolerado pela aplicação. A DFR permite avaliar se o mecanismo de armazenamento em cache dos nós de borda ajuda a aumentar o desempenho do sistema em termos de redução do tempo de resposta das requisições e da redução energia consumida pelos dispositivos da camada das coisas. A métrica NEN é o número total de nós de borda usados para gerar as diferentes topologias e permite avaliar a escalabilidade do sistema. Nos experimentos a seguir, um desses três parâmetros é variado e os dois parâmetros restantes são configurados com o valor de RAR, DFR ou NEN correspondente e que está descrito na Tabela 6.1.

6.3.1 Experimento E1

Nós projetamos o experimento E1 para responder as perguntas Q1, Q2 e Q3, que avaliam o desempenho do sistema em relação à taxa de chegada de requisições (RAR). Durante a execução do experimento, nós configuramos o valor de frescor do dado (DFR) e o número de nós de borda (NEN) com os valores descritos na Tabela 6.1. Além disso, nós variamos o RAR de 5 requisições por segundo até 500 requisições por segundo. Em todos os gráficos da Figura 6.1, o eixo x corresponde ao RAR e o eixo y corresponde às métricas RRT, NMT, PFR e ECT (Tabela 6.3).

Na Figura 6.1(a), todos os cenários mostram o mesmo comportamento de aumento do valor do RRT à medida que o RAR é aumentado. Vale ressaltar que os valores para o cenário Nuvem também apresentaram um aumento. No entanto, para facilitar a visualização dos outros cenários, a curva do cenário Nuvem foi plotada em uma escala diferente (em vermelho) na Figura 6.1(a).

A Figura 6.1(a) mostra que o RTT atingiu seus valores máximos apenas no cenário de duas camadas (cenário Nuvem). Os picos da métrica RRT alcançados no cenário Nuvem geralmente ocorrem porque as requisições das aplicações são atendidas diretamente na nuvem, a qual possui uma latência de comunicação com a camada de coisas mais significativa . Além disso, o menor desempenho do cenário Nuvem ocorre porque as requisições chegam de todas as regiões, enquanto que nos cenários de três camadas, cada nó de borda é responsável por atender apenas requisições de uma área geográfica pré-estabelecida pelo diagrama de Voronoi. Esses fatores apontam para a eficácia de uma camada intermediária entre a camada de coisas e a camada de nuvem.

Nos cenários de três camadas (Vertical, Horizontal e Hierárquico), as curvas apresentam uma queda suave até que o valor do RAR é igual a 10 requisições por segundo. No entanto, quando o RAR é maior do que 10 requisições por segundo, o valor do RRT para o cenário Vertical aumenta, enquanto nas abordagens Horizontal e Hierárquico, o RRT continua caindo até 50 requisições por segundo. No

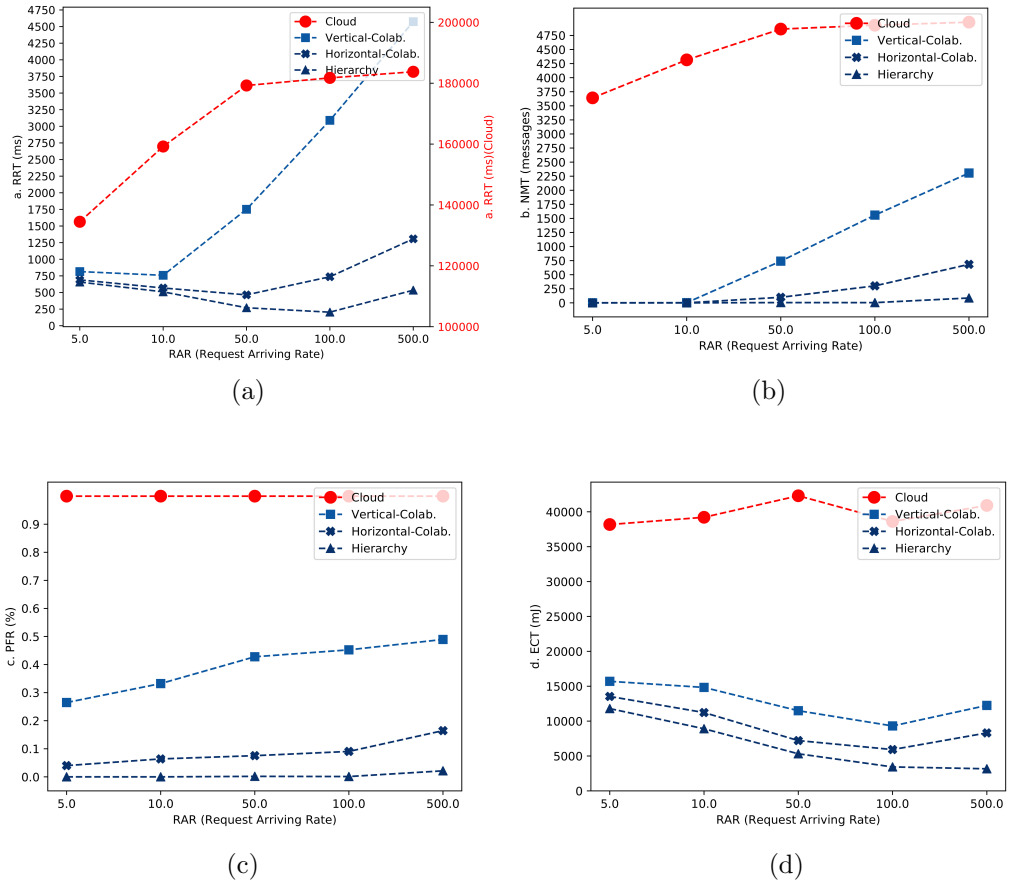


Figura 6.1: Resultados de (a) RRT, (b) NMT, (c) PFR e (d) ECT para diferentes valores de Taxa de Chegada de Requisição (RAR).

cenário Vertical, o aumento do RTT é consideravelmente maior do que nos cenários Horizontal e Hierárquico. Para RAR igual a 100 e 500 requisições por segundo, o RRT do cenário Vertical é de 3089 ms e 4573 ms contra um RRT de 736 ms e 1306 ms para o cenário Horizontal, respectivamente, e RRT de 204 ms e 534 ms para o cenário Hierárquico, respectivamente.

Esse aumento abrupto do RRT no final da curva do cenário Vertical ocorre porque os nós de borda não realizam encaminhamento de requisições dentro da camada de borda, o que faz com que a maioria das requisições sejam encaminhadas para a nuvem. No cenário Vertical, o nó de borda possui apenas uma conexão estabelecida com a nuvem, e o aumento do RAR leva ao consumo de mais recursos (memória e tempo de CPU) dos nós. Esse consumo faz com que as requisições sejam encaminhadas rapidamente para a nuvem, aumentando assim o tempo de resposta para atender as requisições.

O desempenho dos cenários Horizontal e Hierárquico, para RRT > 10, é significativamente maior do que o desempenho para os cenários Vertical e Nuvem, pois nas abordagens Horizontal e Hierárquico as requisições são encaminhadas entre os nós da camada de borda. Nós cenários Horizontal e Hierárquico, os nós de borda têm conexões entre si, o que permite que os nós executem o encaminhamento das requisições entre os nós de borda.

Nos cenários Nuvem e Vertical, os dispositivos da IoT (para o cenário Nuvem) e os nós de borda (para o cenário Vertical) são conectados diretamente à nuvem, respectivamente, o que impede o encaminhamento da requisição na camada de borda. Assim, os cenários Horizontal e Hierárquico superam os cenários Nuvem e Vertical. No entanto, o cenário Hierárquico ainda possui um valor de RRT maior do que o cenário Horizontal. Para RAR igual a 100 e 500 requisições por segundo, o RRT para o cenário Hierárquico apresenta um ganho de 261% e 144% em comparação com o cenário Horizontal, respectivamente.

Alguns fatores explicam esse desempenho superior do cenário Hierárquico. O primeiro fator é a configuração da rede em uma estrutura hierárquica, a qual permite que nós escravos com recursos esgotados possam encaminhar requisições para outros nós até chegar em um nó mestre. O segundo fator é a redução do comprimento do caminho entre os nós de borda no cenário Hierárquico. No cenário Horizontal, os nós são organizados em uma topologia em linha, portanto, a distância entre um nó com recursos esgotados e um nó com recursos disponíveis é a distância máxima possível, ou o pior caso para um grafo que fosse gerado aleatoriamente.

O cenário Hierárquico é construído em uma topologia em árvore para minimizar a distância física entre cada nó escravo e o nó mestre da rede. Portanto, na maioria dos casos, a distância física (caminho) para alcançar um nó com recursos disponíveis será menor. Esse caminho mais curto na camada de borda ajuda a diminuir a latência

para atender à requisição, pois as distâncias entre dois nós de borda podem atingir um intervalo de dezenas ou centenas de metros. Ao reduzir a distância entre os nós na camada de borda, a arquitetura do cenário Hierárquico ajuda a diminuir o valor de RRT.

A Figura 6.1(b) mostra os resultados para o tráfego de mensagens da rede (NMT). Esta figura ajuda a entender os resultados apresentados na Figura 6.1(a). As curvas na Figura 6.1(b) mostram que durante a simulação, um grande número de mensagens (4986 mensagens para RAR igual a 500 requisições por segundo) permaneceu nas filas de processamento entre a camada das coisas e a nuvem no cenário Cloud. Nas abordagens de três camadas, o NMT é aproximadamente constante e próximo de zero até um valor de RAR igual a 10 requisições por segundo. No cenário Vertical, para valores de RAR maiores do que 10, o valor de RRT aumenta abruptamente, o NMT passa de quase zero, para RAR = 10 req./s, para NMT igual a 2306, para RAR = 500 req./s. No cenário Horizontal, o valor de NMT apresenta um pequeno aumento para RAR maior do que 10 req./seg (NMT = 684 mensagens no pior caso) e no cenário Hierárquico, o valor de NMT apresenta uma pequena oscilação para todos os valores do RAR (NMT = 86 mensagens no pior caso). Portanto, os resultados apresentados na Figura 6.1(a) e na Figura 6.1(b) ajudam a responder à pergunta Q1, porque mostram que: há uma redução efetiva do tempo de resposta de requisição nos cenários de três camadas em comparação ao cenário de duas camadas; o tráfego de rede gerado pela troca de mensagens entre os nós de borda e entre a camada de borda e a camada de nuvem é consideravelmente menor em cenários de três camadas; e o algoritmo de alocação de recursos é capaz de se beneficiar da estrutura em árvore do cenário Hierárquico para atender com eficiência as requisições e equilibrar a carga de trabalho entre os nós da camada de borda.

A Figura 6.1(c) mostra os resultados dos valores de PFR, variando os valores de RAR. No cenário Nuvem, (obviamente) 100% das requisições são encaminhadas para a nuvem. Considerando o cenário Vertical, a alta latência durante a comunicação entre os nós de borda e a nuvem aumenta a quantidade de mensagens nas filas (Figura 6.1(b)). Esse comportamento aumenta o número de mensagens encaminhadas para a nuvem. Para o cenário Vertical, o PFR máximo é igual a 49% das requisições. Quando há o encaminhamento de requisição na camada de borda (cenários Horizontal e Hierárquico), os nós de borda com recursos esgotados podem encaminhar as requisições recebidas para os nós de borda vizinhos com recursos disponíveis, evitando assim o crescimento da quantidade de mensagens na fila. No entanto, vale observar que o valor de PRF é sempre menor no cenário Hierárquico do que no cenário Horizontal. Até RAR = 100 req./seg, zero mensagens são encaminhadas para a nuvem no cenário Hierárquico, e para RAR = 500 req./seg, o PRF é apenas igual a 2%. Para o cenário Horizontal, no pior caso, o PRF é igual a 21%.

Portanto, o resultado dos experimentos ajudam a responder à pergunta Q2, porque mostram que o cenário Hierárquico é mais eficiente para evitar o encaminhamento de requisições para a nuvem, além disso, todas as abordagens de três camadas são mais eficientes que o cenário Nuvem.

A Figura 6.1(d) mostra o consumo de energia (ECT) para atender as requisições durante o tempo total da simulação. Os resultados mostram que a energia consumida é aproximadamente constante nos cenários de três camadas. No entanto, quando consideramos o cenário da nuvem, o consumo de energia é sempre maior. É importante observar que no cenário de duas camadas, não há mecanismo de cache, enquanto no cenário de três camadas cada nó de borda possui uma cache para manter os dados obtidos na camada das coisas. Além disso, o frescor do dado é usado para verificar a validade dos dados na cache do nó de borda. Os dados da cache são usados quando o requisito de frescor do dado solicitado corresponde ao tempo de aquisição do dado na cache. Caso o requisito de frescor do dado não seja atendido, o nó de borda acessa o dispositivo da IoT para obter o dado atualizado e atualizar sua cache. No cenário de duas camadas, o nó da nuvem sempre precisa coletar os dados da camada das coisas. Esses resultados ajudam a responder à pergunta Q3. Eles mostram a eficácia do mecanismo de cache do algoritmo de alocação de recursos dos nós de borda. Esse mecanismo de cache evita a troca de mensagens entre os dispositivos da IoT e os nós de borda, fazendo com que o algoritmo de alocação de recursos reduza a energia gasta dos dispositivos da IoT.

6.3.2 Experimento E2

O experimento E2 foi projetado para ajudar a responder as perguntas Q1, Q2 e Q3, avaliando a escalabilidade do sistema por meio da variação do parâmetro número de nós de borda (NEN). Durante a execução do experimento, nós alteramos o número de nós de borda (NEN), variando-o de 10 a 50 e configuramos os valores da taxa de chegada de requisição (RAR) e do requisito de frescor do dado (DFR) para os valores padrão descritos na Tabela 6.1. A Figura 6.2 mostra os resultados para as métricas RRT, NMT, PFR e ECT no eixo y, variando os valores de NEN no eixo x.

A Figura 6.2(a) mostra os resultados da métrica RRT. É importante ressaltar que a curva para o cenário Nuvem está em uma escala diferente. Nós podemos observar que a curva do cenário Nuvem é aproximadamente constante. As curvas dos cenários Vertical e Horizontal possuem o comportamento semelhante, onde o RRT é inversamente proporcional ao NEN. Essa relação entre o RRT e o NEN ocorre pois quanto maior é o número de nós de borda que compõem a camada de borda, maior é a quantidade de recursos (tempo de uso da CPU e memória) disponível e,

principalmente, maior é a probabilidade de algum nó de borda possuir o tipo de dado requisitado pela aplicação. Isso significa que quando há mais nós de borda, maior é a chance da requisição ser atendida na camada de borda e mais requisições podem ser atendidas por unidade de tempo. Por exemplo, o valor máximo de NEN é de 50 nós, o que implica cerca de cinco vezes mais memória e poder de computação espalhados na camada de borda quando comparado ao valor mínimo igual a 10 nós. Isso significa que uma requisição que chega em um nó de borda que não possua o seu tipo de dado provavelmente será encaminhada para algum nó de borda que possua o tipo de dado, além de recursos disponíveis, sendo atendida na camada de borda. Portanto, quanto mais recursos disponíveis e mais tipos de dados oferecidos pelos nós de borda, menor a necessidade das requisições serem encaminhadas para a nuvem.

No cenário Vertical, não há encaminhamento de requisição entre os nós de borda, portanto, quando o número de nós é pequeno (10 e 20 nós), o RRT possui valores altos (5581 ms e 1757 ms). No entanto, o RRT permanece aproximadamente constante quando o NEN possui valores altos (30 a 50 nós), próximos a 710 a 780 ms, porque a quantidade total de recursos disponíveis no nível de borda é maior e, portanto, os nós de borda podem atender a todas as requisições sem necessidade de recorrerem à nuvem. O cenário Horizontal tem comportamento semelhante, mas sua curva diminui de maneira mais suave à medida que o número de nós de borda aumenta. Por exemplo, para 10 nós de borda, o valor de RRT é 1184 ms e se aproxima do valor de 460 ms, para 50 nós de borda, à medida que o número de nós aumenta. O principal fator para o cenário Horizontal superar o cenário Vertical é o processo de encaminhamento de requisição entre os nós na camada de borda.

O cenário Hierárquico apresenta uma curva que inicia com o valor mínimo de RTT de 185 ms, que é mais de 5 vezes superior ao da configuração Horizontal. À medida que o número de nós de borda aumenta, o valor de RRT se mantém aproximadamente constante, em torno de 300-400ms. Esses valores são aproximadamente 16% a 63% superiores aos valores de RRT alcançados no cenário Horizontal. A curva aproximadamente constante para todos os valores de NEN no cenário Hierárquico é diferente dos outros cenários de três camadas. Nós observamos que o principal fator para esse resultado foi o uso da estrutura hierárquica para reduzir a distância entre nós, o que permite maior encaminhamento de requisições. Esse fator permite reduzir o número de requisições encaminhadas para a nuvem (Figura 6.2(c)), o qual também é um fator importante para manter o RRT constante. É importante notar que para valores de NEN maiores que 20 nós, todas as curvas apresentam comportamento aproximadamente constante do RRT, além de que os resultados para os cenários Horizontal e Hierárquico serem similares. O cenário Horizontal e Hierárquico possuem valores semelhantes porque os nós de borda nos dois cenários são capazes

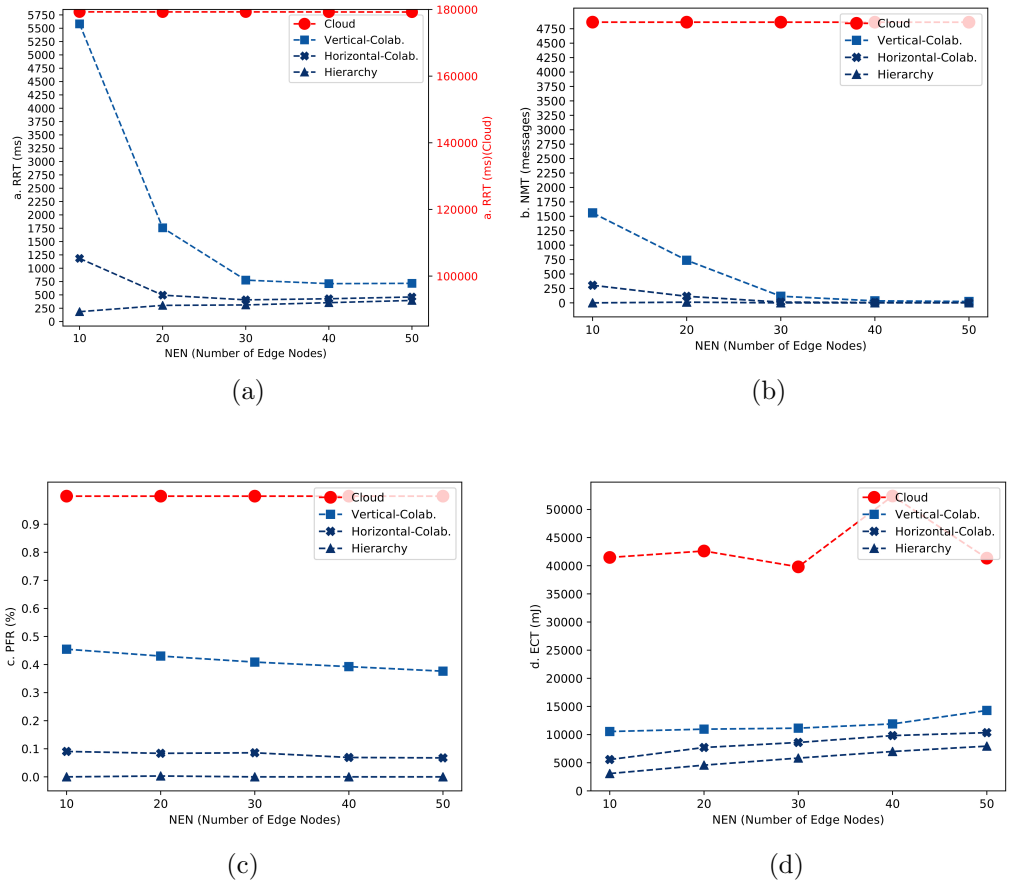


Figura 6.2: Resultados de (a) RRT, (b) NMT, (c) PFR e (d) ECT para diferentes valores de número de nós de borda (NEN).

de atender quase todas as requisições, encaminhando quase zero requisições para a nuvem. Para valores de NEN com mais de 20 nós, a rede possui mais recursos disponíveis e os nós de borda podem atender à maioria das requisições localmente.

A Figura 6.2(b) mostra os resultados obtidos para a métrica NMT. O número total de requisições nas filas de rede é próximo de zero, de 30 a 50 nós de borda, exceto o cenário Nuvem. Nos cenários Vertical e Horizontal, no intervalo de NEN para 10 até 30, o NMT é inversamente proporcional ao NEN. No entanto, o cenário Horizontal apresenta um número menor de mensagens nas filas de rede. O número de mensagens dentro da fila no cenário Hierárquico é sempre próximo de zero, o que explica a curva constante de RRT na Figura 6.2(a) para esse cenário. Esses resultados mostram que a organização da rede em uma estrutura hierárquica gera menos sobrecarga de comunicação e menos mensagens nas filas da rede, incluindo cenários com poucos nós de borda.

Os resultados do RRT (Figura 6.2(a)) e do NMT (Figura 6.2(b)) ajudam a responder à pergunta Q1. Eles mostram que no cenário Hierárquico, o algoritmo de alocação de recursos é bem-sucedido em reduzir o tempo de resposta das requisições e equilibrar a carga de trabalho na camada de borda. Além disso, os cenários Vertical e Horizontal apresentam desempenho satisfatório. Assim, esses resultados sustentam a afirmação de que o algoritmo de alocação de recursos proposto ajuda a reduzir o tempo de resposta da aplicação. Além disso, o cenários de três camadas também são superiores em escalabilidade do que o cenário de duas camadas e a abordagem hierárquica supera os outros cenários para os experimentos realizados.

A Figura 6.2(c) mostra como a redução da troca de mensagens na camada de borda, devido ao aumento de NEN, reduz a porcentagem de requisições encaminhadas para a nuvem (PFR). No cenário vertical, o PFR diminui suavemente de 45% (NEN = 10) para 38% (NEN = 50) à medida que a NEN aumenta. Os cenários Horizontal e Hierárquico superam o Vertical, mas o Hierárquico tem desempenho superior em comparação ao Horizontal. De fato, zero mensagens são encaminhadas para a nuvem para no cenário Hierárquico, enquanto no cenário Horizontal os valores de PFR variam de 7% a 9%. A diferença de desempenho do cenário Vertical para os cenários Horizontal e Hierárquico mostra que as aplicações pagam uma alta penalidade em termos de RRT quando o nó de borda precisa encaminhar muitas requisições (em torno de 38 a 45%) para a nuvem. Além disso, o cenário Hierárquico é o mais eficiente para evitar o encaminhamento de requisições para a nuvem. O cenário Hierárquico possibilita uma maior redução da distância entre os nós da camada de borda e uma maior probabilidade da requisição ser encaminhada para um nó de borda com o tipo de dado requisitado e recursos disponíveis. Esses dois fatores reduzem o número de mensagens aguardando para serem atendidas nas filas, reduzindo assim o PFR. Como resultado, o RRT é menor e a distribuição das re-

quisições entre os nós da rede é superior em comparação aos outros cenários. Esses resultados ajudam a responder à pergunta Q2, pois mostram uma maior redução do PFR nos cenários de três camadas. Para as abordagens de três camadas, o ECT é aproximadamente constante, independentemente do valor de NEN e o cenário Hierárquico supera levemente os cenários Vertical e Horizontal. A Figura 6.2(d) mostra os resultados para o ECT.

6.3.3 Experimento E3

O experimento E3 ajuda a responder as perguntas Q1, Q2 e Q3. A Figura 6.3 mostra o impacto do requisito de frescor do dado (DFR) das requisições de aplicações no desempenho do sistema. Durante a execução do experimento, nós variamos os valores de DFR de 0.001 até 1 segundo e configuramos a taxa de chegada de requisição (RAR) e o número de nós de borda (NEN) para os valores descritos na Tabela 6.1. A Figura 6.3 mostra os resultados das métricas RRT, NMT, PFR e ECT (eixo y) para diferentes valores do requisito de frescor do dado (DFR) no eixo x. No cenário Nuvem, a nuvem possui enormes recursos de memória e poder de computação, mas como não há mecanismo de cache, o nó na nuvem necessita coletar dados da camada das coisas para atender cada requisição que chega. Nos cenários de três camadas, cada nó de borda possui recursos limitados (memória e CPU), mas possui uma cache para armazenar dados recentes coletados dos dispositivos da IoT. Portanto, se o requisito de frescor do dado da requisição corresponder ao valor de frescor de algum dado na cache local do nó de borda, a requisição poderá ser atendida sem a coleta de dados da Camada de Coisas.

Na Figura 6.3(a), a curva do cenário Nuvem foi plotada em uma escala diferente para facilitar a visualização junto com outros cenários. Na Figura 6.3(a), o valor de RRT do cenário Nuvem apresenta um comportamento aproximadamente constante e, nos cenários de três camadas, o valor de RRT é inversamente proporcional ao valor DFR. A curva do cenário Nuvem apresenta baixo desempenho, porque para toda requisição o dado é sempre coletado da camada das coisas. Por outro lado, nos cenários de três camadas, em que há uma cache em cada nó de borda, a curva do RRT é inversamente proporcional ao DFR. Esse comportamento ocorre porque, à medida que o DFR aumenta, a probabilidade do valor de frescor de algum dado mantido em cache corresponder ao valor do requisito de frescor de dado da requisição é maior. Por exemplo, se o valor de DFR é de 1 segundo, há uma probabilidade mais alta de um dado armazenado na cache do nó de borda atender ao requisito de frescor de dado da aplicação, do que para um valor de DFR de 100 milissegundo. Quando se aumenta a probabilidade do frescor de algum dado em cache atender o DFR da requisição, o nó de borda atende uma quantidade maior de requisições sem

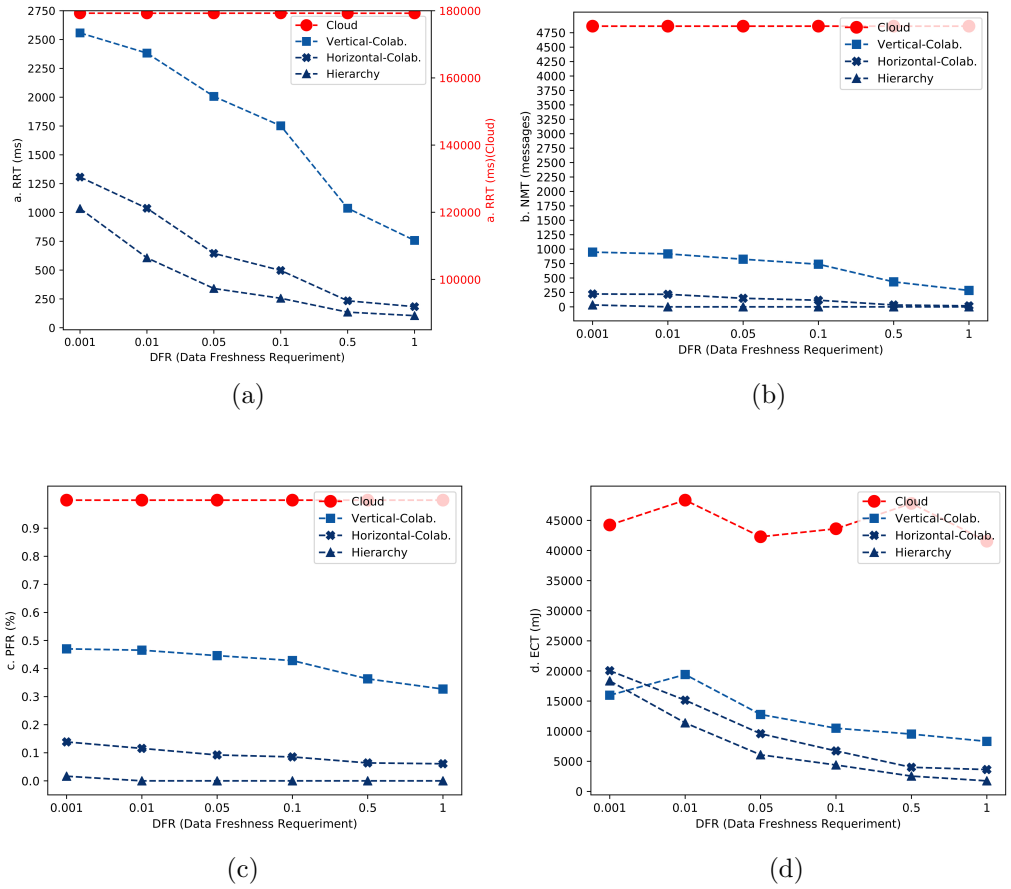


Figura 6.3: Resultados de (a) RRT, (b) NMT, (c) PFR e (d) ECT para diferentes valores de número de nós de borda (NEN).

se comunicar com os dispositivos da IoT na camada das coisas. Isso ocorre porque o dado já está na memória do nó de borda e, consequentemente, o valor do RRT é reduzido.

Na Figura 6.3(a), o cenário Hierárquico apresenta o maior desempenho quando comparado aos outros cenários. De fato, o cenário Hierárquico possui um ganho mínimo de 26%, para o valor de DFR = 0.001, e um ganho máximo de 94%, para DFR = 0.1 segundo, em comparação com o cenário Horizontal. O cenário Hierárquico apresenta esse desempenho, porque permite que o algoritmo de alocação de recursos distribua as requisições com mais eficiência do que nos cenários Vertical ou Horizontal. No entanto, para um alto valor de DFR (1 segundo), o desempenho dos três cenários tende a convergir e o desempenho do cenário Horizontal se aproxima do Hierárquico.

Na Figura 6.3(b), a curva do cenário Nuvem apresenta um comportamento constante e, quando comparado com os cenários de três camadas, seu desempenho é o menor. Considerando apenas os cenários de três camadas, o cenário Vertical apresenta o maior valor de NMT para todos os valores de DFR. A curva do cenário Ho-

rizontal apresenta um número maior de mensagens nas filas de rede para pequenos valores de DFR. No entanto, à medida que o valor de DFR aumenta, principalmente no intervalo de 100 ms a 1 segundo, o valor de NMT do cenário Horizontal se aproxima do valor de NMT do cenário Hierárquico. É importante observar que os baixos valores de NMT, para os cenários Horizontal e Hierárquico, mostram que os nós de borda podem encaminhar requisições exclusivamente na camada de borda com eficiência, reduzindo a quantidade de mensagens nas filas de rede. Essa redução da sobrecarga de comunicação e os resultados do RRT ajudam a responder à pergunta Q1, pois fornecem evidências de que o algoritmo de alocação de recursos pode distribuir de maneira justa as mensagens, mesmo quando as requisições das aplicações possuem um DFR restritivo.

Na Figura 6.3(c), os resultados mostram como a porcentagem de requisições encaminhadas para a nuvem (PFR) foi reduzida nos cenários de três camadas. Obviamente, 100% das requisições no cenário Nuvem foram atendidas na nuvem. Em todos os cenários de três camadas, o PFR sofre uma pequena diminuição à medida que o valor do DFR aumenta. A redução do PFR é explicada, principalmente, em função dos resultados do NMT. O aumento da quantidade de mensagens nas filas de rede impõe atrasos maiores para atender as requisições nos nós de borda, o que gera um aumento dos valores de PRF. Esse resultado ajuda a responder à pergunta Q2, porque mostra a eficácia das abordagens de três camadas em reduzir a porcentagem de requisições encaminhadas para a nuvem.

Na Figura 6.3(d), os resultados mostram uma grande redução do ECT em cenários de três camadas em comparação com a abordagem de duas camadas. O ECT do cenário puramente baseado na nuvem é muito alto e aproximadamente constante, independentemente do DFR, devido à necessidade de sempre os dados serem coletados diretamente da camada das coisas. Os valores de ECT para o cenário Vertical, Horizontal e Hierárquico são inversamente proporcionais ao DFR, o que significa que mais requisições são atendidas com os dados da cache local dos nós de borda. Esse fator ajuda a diminuir a comunicação entre a camada das coisas e a camada de borda. As abordagens de três camadas também reduzem a latência de transmissão de um dispositivo da IoT para um nó de borda, devido à camada das coisas estar mais próxima fisicamente da camada de borda. Por outro lado, no cenário de duas camadas, a nuvem está fisicamente muito distante do dispositivo da IoT, o que implica em maiores latências. O cenário Hierárquico apresenta desempenho superior comparado ao Vertical e Horizontal. De fato, o cenário Hierárquico apresenta uma porcentagem de economia de energia próxima de 107% e 374% em comparação com o cenário Horizontal e Vertical, respectivamente, para DFR = 1 segundo. Esse desempenho ocorre porque o algoritmo de alocação de recursos faz uma distribuição mais equilibrada das requisições na camada de borda. Como pode

ser visto na Figura 6.3(a), o cenário Hierárquico possui o menor valor de RRT em função do DFR, o que significa que, nesse cenário, um nó de borda atende uma quantidade maior de requisições com dados provenientes da cache local. Dessa maneira, é necessário realizar menos comunicação com os dispositivos da IoT e a energia consumida na camada das coisas, medida pelo ECT, é reduzida. Esses resultados ajudam a responder à pergunta Q3, porque sustentam a alegação de que os cenários de três camadas economizam mais energia do que o cenário de duas camadas.

6.3.4 Experimento E4

O experimento E4 foi projetado para ajudar a responder a pergunta Q4 e seus resultados são mostrados na Figura 6.4. A questão Q4 se refere a avaliação da efetividade do algoritmo de alocação de recursos em fornecer uma maior qualidade de serviço, em termos de tempo de resposta da requisição, para requisições com maiores níveis de prioridades. Para responder essa pergunta, nós avaliamos os resultados da métrica RRT-P (eixo y nos gráficos da figura) para diferentes valores dos parâmetros RAR, NEN e DFR, representados no eixo x de cada um dos os três gráficos da Figura 6.4. Neste experimento, nós também variamos o parâmetro que representa o valor financeiro da requisição ψ_{r_k}, pago pelo usuário final que gerou a requisição, ao qual foram atribuídos os valores do conjunto $\{10, 25, 50, 75, 100\}$ (observe que $\Psi = 100$ é definido como o valor financeiro máximo). É importante ressaltar que a prioridade p_{r_k} de uma requisição r_k é diretamente proporcional ao valor financeiro ψ_{r_k} e o cenário considerado foi o Hierárquico.

Na Figura 6.4(a), nós podemos observar que os valores de RRT-P são inversamente proporcionais aos valores de ψ_{r_k}. Esse comportamento para a métrica RRT-P ocorre porque o algoritmo de alocação de recursos estabelece uma probabilidade mais alta para a requisição com maior valor de ψ_{r_k}, para que ela seja atendida rapidamente. Além disso, é importante observar no gráfico que, independentemente do RAR, o RRT-P é sempre proporcional ao valor financeiro (respeitando assim os níveis de prioridades estabelecidos).

Ao analisar os resultados da Figura 6.4(a), e também os resultados do Experimento E1 (ver Figura 6.1(a)), um comportamento interessante do nosso algoritmo pode ser observado para a métrica relacionada ao tempo de resposta da requisição. Para todos os valores de ψ_{r_k}, as curvas da métrica RRT-P apresentam uma queda no intervalo de RAR = [5, 100]. No intervalo de RAR = [100, 500], especialmente para valores altos de ψ_{r_k}, as curvas do RRT-P mudam para um comportamento aproximadamente constante. O comportamento observado no intervalo de RAR = [5, 100] deve-se ao fato de que, a probabilidade de algum dado, armazenado na cache do nó de borda, atender ao requisito de frescor do dado aumenta à medida que o

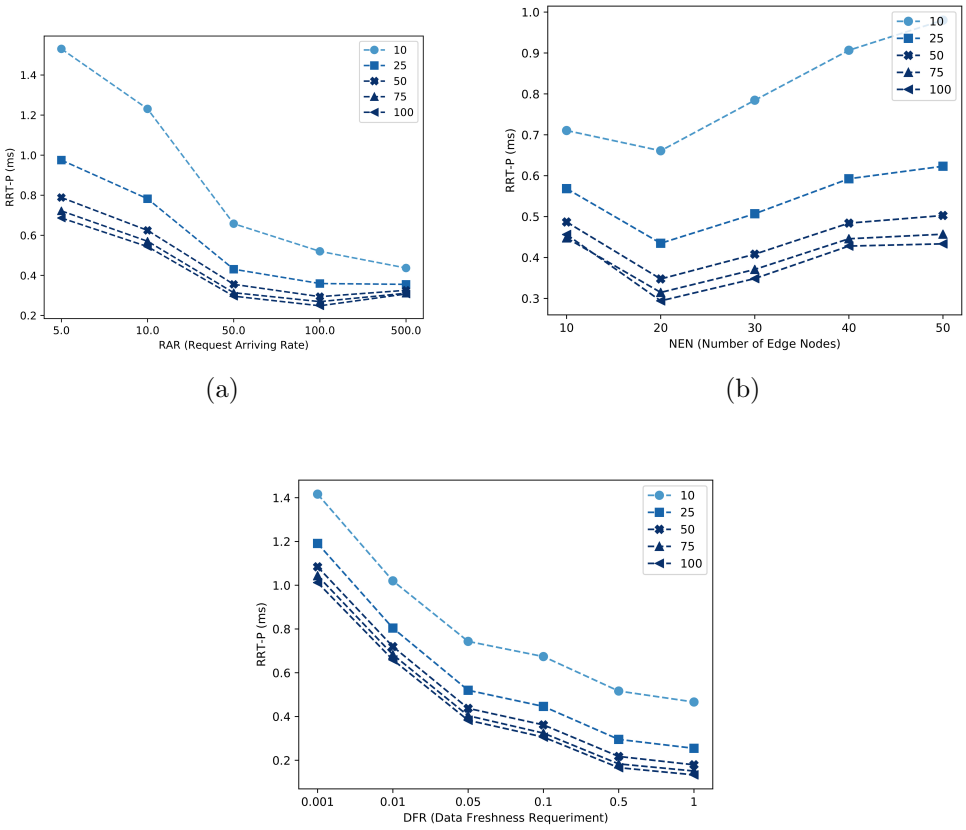


Figura 6.4: Resultados do RRT-P para diferentes valores de (a) RAR, (b) NEN e (c) DFR no cenário Hierárquico.

RAR aumenta.

Para apoiar esta afirmação, nós também coletamos do experimento E4 os valores das seguintes métricas: Porcentagem de Requisições Atendidas com Consulta Local (PFR-LQ - Percentage of Fulfilled Requests with Local Query) e Porcentagem de Requisições Atendidas com Consulta Remota (PFR-RQ - Percentage of Fulfilled Requests with Remote Query) e exibimos os resultados dessas métricas nas duas primeiras colunas da Tabela 6.4. A métrica PFR-LQ corresponde à porcentagem de requisições atendidas, nas quais o dado requisitado foi obtido da cache local do nó de borda. A métrica PFR-RQ corresponde à porcentagem de requisições atendidas, nas quais o dado requisitado foi obtido do dispositivo da IoT na camada de coisas. As métricas PFR-LQ e PFR-RQ correspondem a uma fração do total das requisições enviadas ao sistema, independentemente do valor de prioridade das requisições.

RAR	**PFR-LQ**	**PFR-RQ**	**NMT**
5	35%	65%	0.01
10	49%	51%	0.02
50	74%	25 %	0.10
100	79%	19 %	0.23
500	77%	16%	4.65

Tabela 6.4: Métricas PFR-LQ, PFR-RQ e NMT para diferentes valores de RAR do Experimento E4.

Na Tabela 6.4, podemos observar que os valores de PFR-LQ aumentam no intervalo de RAR = [5, 100], enquanto os valores de PFR-RQ diminuem consideravelmente no intervalo de RAR = [5, 100]. No experimento E4, nós mantivemos o requisito de frescor do dado fixo, logo pode-se concluir que o RAR afeta consideravelmente o PFR-LQ. Isso ocorre porque o aumento do valor de RAR implica em um tempo mais curto entre a chegada de duas requisições consecutivas, o que, por sua vez, implica em um aumento na probabilidade de que algum dado em cache, proveniente de requisições anteriores, seja fresco o suficiente para atender ao requisito de frescor da requisição. Portanto, mesmo se a velocidade de chegada da requisição aumentar, caso o nó de borda possua recursos suficientes, pode ocorrer uma redução no RRT devido ao uso da cache nos nós de borda.

No intervalo de RAR = [100, 500], ocorre uma suavização no comportamento decrescente do RRT-P, pois para esses valores a taxa de chegada da requisições é tão alta que o fator que predomina sobre o RRT-P é a sobrecarga gerada pela comunicação. Na Tabela 6.4, a terceira coluna mostra os valores para a métrica NMT. Podemos observar que, para valores no intervalo de RAR = [5 a 100], o valor de NMT é aproximadamente zero e não oscila significativamente, indicando a quase ausência de filas nos nós de borda. No entanto, para o valor de RAR = 500, o valor de NMT passa para 4,65, que é alto considerando o cenário Hierárquico, e indica

um aumento no número de mensagens nas filas dos nós de borda.

A partir dos resultados da Tabela 6.4 pode-se observar que até o valor de RAR $<$ 100, o fator dominante no comportamento do RRT-P é a porcentagem de requisições que são atendidas direto da cache, sem a coleta do dado na camada das coisas. No entanto, quando RAR $\geq$ 100, o fator dominante que afeta o RRT-P se torna a sobrecarga de comunicação da rede. Essa observação é interessante porque, em muitos cenários reais, a taxa de chegada das requisições não é tão alta quanto 500 requisições / segundo. Portanto, o algoritmo de alocação de recursos pode oferecer desempenho superior, mesmo em cenários em que o valor do RAR aumenta.

A Figura 6.4(b) e a Figura 6.4(c) mostram os valores de RRT-P obtidos variando-se os valores de NEN e DFR, respectivamente. Nos dois gráficos, o RRT-P é inversamente proporcional a ψ_{r_k}. Os resultados mostram que o algoritmo de alocação de recursos é capaz de priorizar requisições de acordo com o valor financeiro atribuído a elas, independentemente do número de nós de borda e do requisito de frescor do dado imposto pela aplicação.

Capítulo 7

Conclusões

Este capítulo apresenta as conclusões e trabalhos futuros para esta tese. Na Seção 7.1, nós tecemos as considerações finais sobre as soluções propostas. Na Seção 7.2, nós apresentamos as questões que ficaram em aberto em nosso trabalho e apontamos alguns trabalhos futuros que podem melhorar ou estender nossas soluções.

7.1 Considerações Finais

Nesta tese, nós apresentamos novas soluções para resolver problemas que causam impacto no desempenho dos serviços de infraestrutura de sistemas emergentes da IoT. Nossa arquitetura de computação para sistemas da IoT é composta de três camadas e nossas soluções são focadas principalmente na camada das coisas e na camada da borda.

Na camada das coisas, nós propusemos um novo algoritmo de sincronização de tempo, o qual é eficiente energeticamente e fornece alta acurácia de sincronização para os nós sensores que compõem a infraestrutura de sensoriamento do sistema da IoT. Nosso algoritmo aborda o problema do erro cumulativo entre os nós sensores, selecionando adequadamente o nó de referência que minimiza a sua distância para os demais nós da rede. A abordagem de seleção proposta faz uso de um temporizador em cada nó sensor, permitindo que o nó sensor estime o diâmetro da rede e a distância dele para o nó sensor mais distante. O mecanismo de seleção baseado em temporizadores permite selecionar o nó de referência da rede sem a necessidade de aumentar o número de mensagens trocadas, pois o processo de seleção utiliza as próprias mensagens de sincronização. Portanto, o algoritmo provê uma alta acurácia da rede, em termos de erro de sincronização, sem a necessidade de aumentar o overhead de comunicação.

Nós avaliamos nosso algoritmo em comparação com uma abordagem do estado da arte, a qual é eficiente energeticamente e não aborda o problema do erro cumulativo. Nós avaliamos o erro de sincronização, mostrando que o algoritmo proposto

proporciona uma maior acurácia do que o algoritmo com o qual foi comparado. A maior acurácia do algoritmo proposto é um resultado importante, pois aplicações críticas da IoT, as quais requerem sincronização e coordenação de eventos dos nós sensores, podem se beneficiar de uma maior acurácia de sincronização de tempo.

O overhead de comunicação do algoritmo de sincronização também foi avaliado, e foi evidenciado que o algoritmo não aumenta a quantidade de mensagens trocadas na rede. Esse resultado, em conjunto com o resultado de aumento da acurácia, pode beneficiar aplicações em cenários em que a economia do consumo de energia é o fator mais crítico, de modo que a acurácia da rede pode ser relaxada.

Os resultados dos experimentos mostraram que o algoritmo de sincronização proposto é capaz de aumentar a acurácia de sincronização da rede sem o aumento de overhead de comunicação. Desse modo, a abordagem é capaz de melhorar a sincronização e coordenação das atividades na camada das coisas, permitindo um uso mais eficiente da infraestrutura física pelas aplicações e pelos demais serviços de infraestrutura que são dependentes da sincronização de tempo dos nós sensores.

Na camada de borda da nuvem das coisas, nós propusemos um algoritmo para abordar o problema de alocação de recursos. Nossa proposta alavanca a computação de borda para prover recursos, os quais são mapeados para nós virtuais, próximos à fonte de dados. Os nós da camada de borda são organizados em um modelo hierárquico e a colaboração entre eles é promovida para distribuir o esforço de atender as requisições das aplicações de uma forma balanceada.

Nós propusemos um algoritmo que organiza os nós da camada de borda em uma estrutura hierárquica, cujo objetivo é reduzir a distância, e consequentemente a latência, entre os nós de borda. A organização hierárquica dos nós de borda ajuda a tratar os desafios de grande escala e alta heterogeneidade dos sistemas da CoT. Por causa da organização dos nós de borda baseada em hierarquia, as requisições são encaminhadas para nós de borda mais próximos.

A partir desta arquitetura, nós propusemos um algoritmo de alocação de recursos que promove a colaboração entre os nós de borda. O algoritmo de alocação de recursos fornece a habilidade para os nós de borda decidirem localmente quando engajar seus vizinhos para alocarem nós virtuais sempre que é necessário atender as requisições das aplicações.

O algoritmo de alocação de recursos proposto é inspirado em um modelo de mercado econômico, que considera a limitação de recursos da infraestrutura física da CoT para atender as aplicações. O algoritmo fornece uma função de utilidade para estimar o custo do nó de borda em fornecer algum recurso e o preço pago pelo usuário para utilizar tal recurso. Quando o nó de borda não é capaz de atender uma requisição, nosso algoritmo encaminha a requisição para o nó de borda vizinho com as melhores condições para atender as aplicações, permitindo uma distribuição

homogênea dos recursos da infraestrutura. A orquestração dos recursos e o encaminhamento das requisições permite que o algoritmo realize um balanceamento de carga de trabalho e atenda os requisitos de latência das aplicações. O compartilhamento de requisições, não apenas aprimora o uso dos recursos da infraestrutura, como também tem o efeito de mover menos requisições para a nuvem.

O algoritmo proposto considera simultaneamente os requisitos de tempo de resposta e frescor do dado em sua formulação. Além disso, ele permite que diferentes valores de prioridades sejam definidos para diferentes aplicações, a fim de fornecer qualidade de serviço para as aplicações com maior grau de criticidade, acomodando assim adequadamente os requisitos das aplicações no complexo ecossistema da CoT.

Os resultados dos experimentos realizados mostraram que nosso algoritmo supera uma arquitetura da CoT de duas camadas, baseada em nuvem, além dos outros cenários avaliados, em termos de atender mais requisições de aplicações, diminuindo o tempo de resposta, economizando energia dos dispositivos da IoT e reduzindo o tráfego geral de dados gerado no sistema. O compartilhamento de dados habilitou a manutenção de dados de sensoriamento armazenados em caches locais na camada de borda, aumentando a capacidade do sistema em atender um maior número de aplicações sem a necessidade de coletar mais dados dos dispositivos da IoT.

7.2 Trabalhos Futuros

Com relação ao algoritmo de sincronização de tempo, embora os resultados tenham demonstrado sua efetividade em atingir os objetivos de aumento da acurária e eficiência energética em comparação com o estado da arte, nós escolhemos alguns pontos a serem explorados em trabalhos futuros.

- O esquema de seleção de nó referência por meio de temporizadores pode ser estendido para outros protocolos de sincronização em que a acurácia seja o fator crítico.

- O algoritmo de seleção de nó de referência pode ser estudado em estratégias que tratam conjuntamente o problema de sincronização de tempo e problema de localização. Muitos problemas de localização de RSSFs necessitam escolher um nó que agregue os dados da rede de nós sensores, de maneira que uma abordagem conjunta como a mencionada acima pode resolver os dois problemas eficientemente.

- Nós avaliamos o algoritmo de sincronização de tempo em um ambiente de simulação, no entanto o algoritmo pode ser implementado em um plataforma de nós sensores a fim de avaliar sua aplicabilidade em um ambiente real.

- Embora os nós de borda geralmente sejam ligados a uma fonte de energia, o consumo energético é uma métrica que se deseja reduzir em todo o sistema da CoT. Além disso, a sincronização dos nós da camada de borda é necessária principalmente para avaliação de métricas de rede multi-hops. Dessa forma, o algoritmo de sincronização proposto pode ser abordado como um mecanismo eficiente para sincronizar os nós da camada de borda também.

Com relação a nossa proposta de gerenciamento de recursos na nuvem das coisas, nós apontamos algumas limitações que podem ser abordadas em trabalhos futuros.

- O algoritmo de alocação de recursos permite a colaboração de um nó com seu vizinho, sem que o nó necessite conhecer *a priori* os tipos de dados do nó vizinho. Tendo esta característica em vista, o algoritmo pode ser avaliado em ambientes móveis e dinâmicos da camada das coisas, já que a mudança dinâmica dos tipos de dados do nó de borda não impede que o nó de borda possa realizar a colaboração com seu vizinho.

- Um dos principais motivos para o aumento do tempo de resposta da aplicação é o aumento das filas nos nós de borda. Tais filas aumentam de tamanho principalmente porque os nós de borda não possuem o tipo de dado necessário para atender a aplicação. A fim de reduzir o tamanho das filas dos nós de borda, esquemas de gerenciamento de filas e compartilhamento de requisições em fila podem ser abordados em conjunto com o algoritmo de alocação proposto.

- Nosso modelo de virtualização assume que os tipos de dados são definidos *a priori* pelo fornecedor de infraestrutura. No entanto, a definição do tipo de dados pode ser realizada dinamicamente, de acordo com as características das aplicações. Tal abordagem requer o estudo de uma linguagem para definição dos tipos de dados para mapear a semântica do tipo de dado para a sua implementação na infraestrutura.

- O mecanismo de colaboração pode realizar o encaminhamento de requisições para nós vizinhos de maneira mais inteligente, antecipando a decisão de encaminhamento. Quando um nó consulta seu vizinho para avaliar a possibilidade de encaminhar a requisição para ele, o nó vizinho passa a saber sobre o tipo de dado do nó que realizou a consulta. Na chegada de nova requisição no nó vizinho, ele pode encaminhá-la antecipadamente para o nó que realizou a consulta anteriormente.

Referências Bibliográficas

ALI, M., RIAZ, N., ASHRAF, M. I., et al., 2018, "Joint Cloudlet Selection and Latency Minimization in Fog Networks", *IEEE Transactions on Industrial Informatics*, v. 14, n. 9 (Sep.), pp. 4055–4063. ISSN: 1941-0050. doi: 10.1109/TII.2018.2829751.

ALRAWAHI, A. S., LEE, K., LOTFI, A., 2019, "A Multiobjective QoS Model for Trading Cloud of Things Resources", *IEEE Internet of Things Journal*, v. 6, n. 6 (Dec), pp. 9447–9463. ISSN: 2372-2541. doi: 10.1109/JIOT.2019.2943284.

ALVAREZ-CAMPANA, M., LÓPEZ, G., VÁZQUEZ, E., et al., 2017, "Smart CEI Moncloa: An IoT-based Platform for People Flow and Environmental Monitoring on a Smart University Campus", *Sensors*, v. 17 (12), pp. 2856. doi: 10.3390/s17122856.

BASILI, V. R., 1992, *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. Relatório técnico, University of Maryland at College Park, USA.

BENZAÏD, C., BAGAA, M., YOUNIS, M., 2014, "An efficient clock synchronization protocol for wireless sensor networks". In: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 718–723, Aug. doi: 10.1109/IWCMC.2014.6906444.

BENZAÏD, C., BAGAA, M., YOUNIS, M., 2017, "Efficient clock synchronization for clustered wireless sensor networks", *Ad Hoc Networks*, v. 56, pp. 13 – 27. ISSN: 1570-8705. doi: https://doi.org/10.1016/j.adhoc.2016.11.003. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1570870516302992>.

BITAM, S., ZEADALLY, S., MELLOUK, A., 2018, "Fog computing job scheduling optimization based on bees swarm", *Enterprise Information Systems*, v. 12, n. 4, pp. 373–397. doi: 10.1080/17517575.2017.1304579. Disponível em: <https://doi.org/10.1080/17517575.2017.1304579>.

BOUZEGHOUB, M., 2004, "A Framework for Analysis of Data Freshness". In: *Proceedings of the 2004 International Workshop on Information Quality in Information Systems*, IQIS '04, p. 59–67, New York, NY, USA, JAN. Association for Computing Machinery. ISBN: 1581139020. doi: 10.1145/1012453.1012464. Disponível em: <https://doi.org/10.1145/1012453.1012464>.

CHALAPATHI, G. S. S., MANEKAR, R., CHAMOLA, V., et al., 2016, "Hardware validated efficient and simple Time Synchronization protocol for clustered WSN". In: *2016 IEEE Region 10 Conference (TENCON)*, pp. 2162–2166, Nov. doi: 10.1109/TENCON.2016.7848409.

CHALAPATHI, G. S. S., CHAMOLA, V., GURUNARAYANAN, S., et al., 2019, "E-SATS: An Efficient and Simple Time Synchronization Protocol for Cluster- Based Wireless Sensor Networks", *IEEE Sensors Journal*, v. 19, n. 21 (Nov), pp. 10144–10156. ISSN: 2379-9153. doi: 10.1109/JSEN.2019.2922366.

CHANG, K.-D., CHEN, C.-Y., CHEN, J.-L., et al., 2011, "Internet of Things and Cloud Computing for Future Internet". In: Chang, R.-S., Kim, T.-h., Peng, S.-L. (Eds.), *Security-Enriched Urban Computing and Smart Grid*, pp. 1–10, Berlin, Heidelberg, JAN. Springer Berlin Heidelberg. ISBN: 978-3-642-23948-9.

DEHURY, C. K., SAHOO, P. K., 2016, "Design and implementation of a novel service management framework for IoT devices in cloud", *Journal of Systems and Software*, v. 119, pp. 149 – 161. ISSN: 0164-1212. doi: https://doi.org/10.1016/j.jss.2016.06.059. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121216300887>.

DELICATO, F. C., PIRES, P. F., BATISTA, T., 2017, In: *Resource Management for Internet of Things*, 1st ed., Springer Publishing Company, Incorporated. ISBN: 331954246X.

DENG, R., LU, R., LAI, C., et al., 2016, "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption", *IEEE Internet of Things Journal*, v. 3, n. 6 (Dec), pp. 1171–1181. ISSN: 2372-2541. doi: 10.1109/JIOT.2016.2565516.

DJENOURI, D., MERABTINE, N., MEKAHLIA, F. Z., et al., 2013, "Fast Distributed Multi-Hop Relative Time Synchronization Protocol and Estimators for Wireless Sensor Networks", *Ad Hoc Netw.*, v. 11, n. 8 (nov.),

pp. 2329–2344. ISSN: 1570-8705. doi: 10.1016/j.adhoc.2013.06.001. Disponível em: <https://doi.org/10.1016/j.adhoc.2013.06.001>.

DO, C. T., TRAN, N. H., CHUAN PHAM, et al., 2015, "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing". In: *2015 International Conference on Information Networking (ICOIN)*, pp. 324–329, Jan. doi: 10.1109/ICOIN.2015.7057905.

ELSON, J., GIROD, L., ESTRIN, D., 2003, "Fine-Grained Network Time Synchronization Using Reference Broadcasts", *SIGOPS Oper. Syst. Rev.*, v. 36, n. SI (dez.), pp. 147–163. ISSN: 0163-5980. doi: 10.1145/844128.844143. Disponível em: <https://doi.org/10.1145/844128.844143>.

GANERIWAL, S., KUMAR, R., SRIVASTAVA, M. B., 2003, "Timing-Sync Protocol for Sensor Networks". In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, p. 138–149, New York, NY, USA, JAN. Association for Computing Machinery. ISBN: 1581137079. doi: 10.1145/958491.958508. Disponível em: <https://doi.org/10.1145/958491.958508>.

HE, L., 2008, "Time Synchronization Based on Spanning Tree for Wireless Sensor Networks". In: *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4, Oct. doi: 10.1109/WiCom.2008.846.

JAIN, S., SHARMA, Y., 2011, "Optimal Performance Reference Broadcast Synchronization (OPRBS) for time synchronization in wireless sensor networks". In: *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*, pp. 171–175, March. doi: 10.1109/ICCCET.2011.5762462.

JIA, X., LU, Y., WEI, X., et al., 2019, "Improved Time Synchronization Algorithm for Wireless Sensor Networks based on Clustering". In: *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pp. 1211–1215, May. doi: 10.1109/ITAIC.2019.8785426.

JIEHAN ZHOU, LEPPANEN, T., HARJULA, E., et al., 2013, "CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing". In: *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 651–657, June. doi: 10.1109/CSCWD.2013.6581037.

LENKA, R. K., RATH, A. K., TAN, Z. S., et al., 2018, "Building Scalable Cyber-Physical-Social Networking Infrastructure Using IoT and Low Power Sensors", *IEEE Access*, v. 6, pp. 30162–30173.

LENZEN, C., LOCHER, T., WATTENHOFER, R., 2008, "Clock Synchronization with Bounded Global and Local Skew". In: *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 509–518, Oct. doi: 10.1109/FOCS.2008.10.

LENZEN, C., SOMMER, P., WATTENHOFER, R., 2015, "PulseSync: An Efficient and Scalable Clock Synchronization Protocol", *IEEE/ACM Transactions on Networking*, v. 23, n. 3 (June), pp. 717–727. ISSN: 1558-2566. doi: 10.1109/TNET.2014.2309805.

LERA, I., GUERRERO, C., JUIZ, C., 2019, "YAFS: A Simulator for IoT Scenarios in Fog Computing", *IEEE Access*, v. 7, pp. 91745–91758. ISSN: 2169-3536. doi: 10.1109/ACCESS.2019.2927895.

LI, W., SANTOS, I., DELICATO, F. C., et al., 2017, "System modelling and performance evaluation of a three-tier Cloud of Things", *Future Generation Computer Systems*, v. 70, pp. 104 – 125. ISSN: 0167-739X. doi: https://doi.org/10.1016/j.future.2016.06.019. Disponível em: <`http://www.sciencedirect.com/science/article/pii/S0167739X16302047`>.

LI, X., MA, Q., SUN, W., et al., 2014, "Topology shaping for time synchronization in wireless sensor networks". In: *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 33–40, Dec. doi: 10.1109/PADSW.2014.7097788.

LI, X., LIAN, Z., QIN, X., et al., 2018, "Topology-Aware Resource Allocation for IoT Services in Clouds", *IEEE Access*, v. 6, pp. 77880–77889. ISSN: 2169-3536. doi: 10.1109/ACCESS.2018.2884251.

LIM, R., MAAG, B., THIELE, L., 2016, "Time-of-Flight Aware Time Synchronization for Wireless Embedded Systems". In: *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16, p. 149–158, USA, JAN. Junction Publishing. ISBN: 9780994988607.

MARÓTI, M., KUSY, B., SIMON, G., et al., 2004, "The Flooding Time Synchronization Protocol". In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, p. 39–49, New York, NY, USA, JAN. Association for Computing Machinery. ISBN:

1581138792. doi: 10.1145/1031495.1031501. Disponível em: <https://doi.org/10.1145/1031495.1031501>.

MILLS, D. L., 1991, "Internet time synchronization: the network time protocol", *IEEE Transactions on Communications*, v. 39, n. 10 (Oct), pp. 1482–1493. ISSN: 1558-0857. doi: 10.1109/26.103043.

MULLEN, T., WELLMAN, M. P., 1996, "Some issues in the design of market-oriented agents". In: Wooldridge, M., Müller, J. P., Tambe, M. (Eds.), *Intelligent Agents II Agent Theories, Architectures, and Languages*, pp. 283–298, Berlin, Heidelberg, JAN. Springer Berlin Heidelberg. ISBN: 978-3-540-49594-9.

NAYYAR, A., PURI, V., 2016, "A review of Arduino board's, Lilypad's Arduino shields". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1485–1492, March.

NOH, K., SERPEDIN, E., 2007, "Pairwise Broadcast Clock Synchronization for Wireless Sensor Networks". In: *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1–6, June. doi: 10.1109/WOWMOM.2007.4351793.

NOH, K.-L., WU, Y.-C., QARAQE, K., et al., 2008, "Extension of Pairwise Broadcast Clock Synchronization for Multicluster Sensor Networks", *EURASIP J. Adv. Signal Process*, v. 2008 (jan.). ISSN: 1110-8657. doi: 10.1155/2008/286168. Disponível em: <https://doi.org/10.1155/2008/286168>.

QIU, T., ZHANG, Y., QIAO, D., et al., 2018, "A Robust Time Synchronization Scheme for Industrial Internet of Things", *IEEE Transactions on Industrial Informatics*, v. 14, n. 8 (Aug), pp. 3570–3580. ISSN: 1941-0050. doi: 10.1109/TII.2017.2738842.

QIU, T., CHI, L., GUO, W., et al., 2015, "STETS: A novel energy-efficient time synchronization scheme based on embedded networking devices", *Microprocessors and Microsystems*, v. 39, n. 8, pp. 1285 – 1295. ISSN: 0141-9331. doi: https://doi.org/10.1016/j.micpro.2015.07.006. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0141933115001052>.

SANTOS, I. L., PIRMEZ, L., DELICATO, F. C., et al., 2015, "Olympus: The Cloud of Sensors", *IEEE Cloud Computing*, v. 2, n. 2 (Mar), pp. 48–56. ISSN: 2372-2568. doi: 10.1109/MCC.2015.43.

SANTOS, I. L., PIRMEZ, L., DELICATO, F. C., et al., 2019, "Zeus: A resource allocation algorithm for the cloud of sensors", *Future Generation Computer Systems*, v. 92, pp. 564 – 581. ISSN: 0167-739X. doi: https://doi.org/10.1016/j.future.2018.03.026. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X17312761>.

SHEKHAR, S., GOKHALE, A., 2017, "Enabling IoT Applications via Dynamic Cloud-Edge Resource Management: Poster Abstract". In: *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, p. 331–332, New York, NY, USA, JAN. Association for Computing Machinery. ISBN: 9781450349666. doi: 10.1145/3054977.3057307. Disponível em: <https://doi.org/10.1145/3054977.3057307>.

SINALGO, 2020. "Simulador de Eventos Discretos Sinalgo". https://github.com/sinalgo. Accessed: 2020-02-20.

SKOG, I., HANDEL, P., 2010, "Synchronization by Two-Way Message Exchanges: Cramér-Rao Bounds, Approximate Maximum Likelihood, and Offshore Submarine Positioning", *IEEE Transactions on Signal Processing*, v. 58, n. 4 (April), pp. 2351–2362. ISSN: 1941-0476. doi: 10.1109/TSP.2010.2040669.

VASCONCELOS, D. R., ANDRADE, R. M. C., SEVERINO, V., et al., 2019, "Cloud, Fog, or Mist in IoT? That Is the Question", *ACM Trans. Internet Technol.*, v. 19, n. 2 (mar.). ISSN: 1533-5399. doi: 10.1145/3309709. Disponível em: <https://doi.org/10.1145/3309709>.

WANG, N., MATTHAIOU, M., NIKOLOPOULOS, D. S., et al., 2018. "DYVERSE: DYnamic VERtical Scaling in Multi-tenant Edge Environments". .

WEBB, J., HUME, D., 2018, "Campus IoT collaboration and governance using the NIST cybersecurity framework". In: *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pp. 1–7, March. doi: 10.1049/cp.2018.0025.

XU, J., PALANISAMY, B., LUDWIG, H., et al., 2017, "Zenith: Utility-Aware Resource Allocation for Edge Computing". In: *2017 IEEE International Conference on Edge Computing (EDGE)*, pp. 47–54, June. doi: 10.1109/IEEE.EDGE.2017.15.

YOUSEFPOUR, A., ISHIGAKI, G., GOUR, R., et al., 2018, "On Reducing IoT Service Delay via Fog Offloading", *IEEE Internet of Things Journal*, v. 5, n. 2 (April), pp. 998–1010. ISSN: 2372-2541. doi: 10.1109/JIOT.2017.2788802.

ZHANG, H., XIAO, Y., BU, S., et al., 2017, "Computing Resource Allocation in Three-Tier IoT Fog Networks: A Joint Optimization Approach Combining Stackelberg Game and Matching", *IEEE Internet of Things Journal*, v. 4, n. 5 (Oct), pp. 1204–1215. ISSN: 2372-2541. doi: 10.1109/JIOT.2017.2688925.