MÁRCIO LUIZ FERREIRA MIGUEL

ARQUITETURA SDN PARA REDES DE SENSORES SEM FIO 6LOWPAN

Tese apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Doutor em Informática.

CURITIBA

2018

MÁRCIO LUIZ FERREIRA MIGUEL

ARQUITETURA SDN PARA REDES DE SENSORES SEM FIO 6LOWPAN

Tese apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Doutor em Informática.

Área de Concentração: *Ciência da Computação*
Orientador: Prof. Dr. Manoel Camillo O. Penna Neto

CURITIBA
2018

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

Miguel, Márcio Luiz Ferreira
M636a Arquitetura SDN para redes de sensores sem fio 6LOWPAN / Márcio Luiz
2018 Ferreira Miguel ; orientador: Manoel Camillo O. Penna Neto. – 2018.
127 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2018
Bibliografia: f. 119-127

1. Redes de sensores sem fio. 2. Redes sensoriais. 3. Sistemas de baixa tensão. 4. Rede definida por software (Tecnologia de redes de computadores) I. Penna Neto, Manoel Camillo O. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 22. ed. – 004.65

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós Graduação em Informática

PUCPR
GRUPO MARISTA

ATA DE SESSÃO PÚBLICA

DEFESA DE TESE DE DOUTORADO Nº 52/2017

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGIa
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

Em sessão pública realizada às 14h00 de 12 de Março de 2018, no Auditório Guglielmo Marconi – Bloco 8, ocorreu a defesa da tese de doutorado intitulada **"Arquitetura SDN para Redes de Sensores sem Fio 6LoWPAN"** elaborada pelo aluno **Marcio Luiz Ferreira Miguel**, como requisito parcial para a obtenção do título de **Doutor em Informática**, na área de concentração **Ciência da Computação**, perante a banca examinadora composta pelos seguintes membros:

Prof. Dr. Manoel Camillo de Oliveira Penna Neto (orientador) - PUCPR

Prof. Dr. Marcelo Eduardo Pellenz – PUCPR

Prof. Dr. Edgard Jamhour - PUCPR

Prof. Dr. Mauro Sérgio Pereira Fonseca - UTFPR

Prof.ª Drª. Ana Cristina B. Kochem Vendramin – UTFPR

Após a apresentação da tese pelo aluno e correspondente arguição, a banca examinadora emitiu o seguinte parecer sobre a tese:

Membro	Parecer	
Prof. Dr. Manoel Camillo de Oliveira Penna Neto	(X) Aprovada	( ) Reprovada
Prof. Dr. Marcelo Eduardo Pellenz	(X) Aprovada	( ) Reprovada
Prof. Dr. Edgard Jamhour	(X) Aprovada	( ) Reprovada
Prof. Dr. Mauro Sérgio Pereira Fonseca	(X) Aprovada	( ) Reprovada
Prof.ª Drª. Ana Cristina B. Kochem Vendramin	(X) Aprovada	( ) Reprovada

Portanto, conforme as normas regimentais do PPGIa e da PUCPR, a tese foi considerada:

(X) **APROVADO**
(aprovação condicionada ao atendimento integral das correções e melhorias recomendadas pela banca examinadora, conforme anexo, dentro do prazo regimental)

( ) **REPROVADO**

E, para constar, lavrou-se a presente ata que vai assinada por todos os membros da banca examinadora. Curitiba, 12 de Março de 2018.

Prof. Dr. Manoel Camillo de Oliveira Penna Neto

Prof. Dr. Marcelo Eduardo Pellenz

Prof. Dr. Edgard Jamhour

Prof. Dr. Mauro Sérgio Pereira Fonseca

Prof.ª Drª. Ana Cristina B. Kochem Vendramin

Dedico esta tese à minha mãe, Eleni Andriolas,
que sempre me apoiou na busca de meus sonhos.

Agradecimentos

Meus profundos agradecimentos à todas pessoas que me apoiaram no desenvolvimento desse trabalho, especialmente:

- À minha querida esposa, Renate Dela Bruna, pela paciência nos momentos de desânimo e por se privar do tempo de lazer junto a mim, me ajudando na correção do texto.
- Ao meu orientador de longa data, desde o meu Mestrado, o prof. Dr. Manoel Camillo Penna, pelo direcionamento correto dos trabalhos, por sua dedicação e pelo incentivo constante;
- Aos meus professores do PPGIa, Dr. Edgard Jamhour e Dr. Marcelo Eduardo Pellenz pelas longas conversas que levaram ao amadurecimento da tese;
- Aos gestores da COPEL, pelo programa de incentivo ao desenvolvimento de seus profissionais, cuja bolsa de estudos e a liberação de horas me permitiram a conclusão desse trabalho;
- Aos meus colegas de departamento na COPEL, pelo auxílio às minhas atividades profissionais durante os períodos de estudo;
- Ao prof. Dr. Christian Esteve Rothenberg, por seu trabalho inspirador em Redes Definidas por Software e por seu auxílio na definição do tema da tese;
- Às minhas amigas Cristiane Garbin Langner e Simone Crocetti pelo apoio nas horas certas e por partilharem comigo sua experiência na área acadêmica;
- À prof. Dra. Ana Cristina B. Kochem Vendramin e ao prof. Dr. Mauro Sérgio Pereira Fonseca pela correção do texto e pelas sugestões de melhoria apresentadas durante a defesa da tese.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

6LBR	*6LoWPAN Border Router*
6LoWPAN	*IPv6 Over Low Power Wireless Personal Area Networks*
AMI	*Advanced Metering Infrastructure*
AODV	*Ad hoc On-Demand Distance Vector Protocol*
API	*Application Programming Interface*
ASIC	*Application Specific Integrated Circuits*
CoAP	*Constrained Application Protocol*
CoRE	*Constrained RESTful Environments*
CSMA/CA	Carrier *Sense Multiple Access with Collision Avoidance*
CTP	*Collection Tree Protocol*
DNS	*Domain Name System*
DODAG	*Destination Oriented Directed Acyclic Graph*
DSDV	*Destination-Sequenced Distance Vector Protocol*
DSSS	*Direct Sequence Spread Spectrum*
DTLS	*Datagram Transport Layer Security*
ETX	*Expected Transmission Count*
EWMA	*Exponentially Weighted Moving Average*
HTTP	*Hypertext Transfer Protocol*
IANA	*Internet Assigned Numbers Authority*
IEEE	*Institute of Electrical and Electronics Engineers*
IoT	*Internet of Things*
IPHC	*IP Header Compression*

ISM	*Industrial, Scientific and Medical*
JSON	*JavaScript Object Notation*
LAN	*Local Area Network*
LBR	*LoWPAN Border Router*
LLN	*Low Power and Lossy Network*
LOAD	*6LoWPAN Ad Hoc On-Demand Distance Vector Routing*
LOADng	*Lightweight On-demand Ad hoc Distance-vector Routing*
LTE	*Long Term Evolution*
M2M	*Machine-to-machine*
MAC	*Media Access Control*
MOP	*Mode of Operation*
MP2P	*Multipoint-to-point*
MQTT	*Message Queue Telemetry Transport*
NAN	*Neighbor Area Network*
NHC	*Next-header Compression*
NUD	*Neighbor Unreachability Detection*
ONF	*Open Networking Foundation*
O-QPSK	*Offset-Quadrature Phase-Shift Keying*
OXM	*OpenFlow Extensible Match*
PDR	*Packet Delivery Ratio*
PHY	*Physical Layer*
PLC	*Power Line Communication*
PRR	*Packet Reception Ratio*
RDC	*Radio Duty Cycling*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RPL	*Routing Protocol for Low Power and Lossy Networks*
RSSF	Rede de Sensores sem Fio
RSSI	*Received Signal Strength Indicator*
SD6WSN	*Software Defined 6LoWPAN Wireless Sensor Network*
SD6WSNP	*Software Defined 6LoWPAN Wireless Sensor Network Protocol*
SDN	*Software Defined Networking*

SDWSN	*Software Defined Wireless Sensor Network*
SFD	*Start of Frame Delimiter*
SOC	*System on a Chip*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UART	*Universal Asynchronous Receiver/Transmitter*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
WSN	*Wireless Sensor Network*
WPAN	*Wireless Personal Area Network*
XML	*Extensible Markup Language*

Resumo

Esse trabalho propõe um *framework* que possibilita a aplicação do paradigma SDN (*Software Defined Networking*) em RSSFs (Redes de Sensores Sem Fio) do tipo 6LoWPAN (*IPv6 over Low Power Wireless Personal Area Networks*), constituído de uma arquitetura e de um protocolo de controle de roteamento SDN. Na arquitetura proposta, a decisão de roteamento dos dados transmitidos pelas aplicações de sensoriamento residentes nos nós (plano de dados) é efetuada pelas aplicações de rede, que determinam os caminhos mais adequados para cada tipo de fluxo dentro da RSSF de acordo com suas especificidades. As aplicações utilizam a visão unificada da RSSF mantida por um controlador, que obtém as informações da RSSF interagindo com os seus nós. As decisões de roteamento são materializadas em entradas de tabelas de fluxos instaladas nos nós através de trocas de mensagem entre o controlador e os nós. A troca de mensagens entre o controlador e os nós para manutenção da visão unificada de rede e para a manutenção das entradas nas tabelas de fluxo (plano de controle) é realizada pelo protocolo SD6WSNP (*Software Defined 6LoWPAN Wireless Sensor Network Protocol*), proposto nesse trabalho. O SD6WSNP é baseado no protocolo de camada de aplicação CoAP (*Constrained Application Protocol*), que por sua vez utiliza a pilha IP/UDP com roteamento RPL (*Routing Protocol for Low Power and Lossy Networks*). O encaminhamento dos pacotes no plano de dados é realizado por agentes instalados nos dispositivos sem fio, com base nas informações contidas nas tabelas de fluxo. Por meio de mensagens do SD6WSNP, os agentes recebem as informações de encaminhamento dos fluxos de dados, enviam informações sobre a conectividade do nó (vizinhos e qualidade dos enlaces), parâmetros físicos do sinal de rádio e características dos pacotes de dados recebidos para os quais não existem correspondência na tabela de fluxos. A escolha dos caminhos dentro da RSSF é realizada por aplicações ligadas ao controlador, que com base nas informações coletadas e das restrições de cada nó, estabelece a melhor opção para cada fluxo de dados. Para a validação da proposta, os programas para os nós foram desenvolvidos em linguagem “C” na plataforma Contiki e a aplicação controladora

desenvolvida na linguagem Node.js, em ambiente Linux. O *framework* SD6WSN foi validado por simulação em dois cenários de aplicação típicos para RSSFs, em uma AMI (*Advanced Metering Infrastructure*) e em um cenário com topologia do tipo "grade". As simulações da rede AMI foram realizadas no ambiente Contiki/COOJA, onde comparou-se o desempenho de uma aplicação que calcula os caminhos mínimos entre o 6LBR e os nós da RSSF através de duas métricas de desempenho, taxa de recebimento de pacotes e latência. No cenário com topologia "grade" foi comparada a latência entre pares de nós pertencentes à mesma RSSF para rotas definidas por uma aplicação SD6WSN em relação às definidas pelo RPL. Avaliou-se também o *overhead* introduzido pelo plano de controle SD6WSN, medindo-se a proporção entre a quantidade de pacotes transmitidos nas simulações apenas com o RPL em comparação com os pacotes transmitidos nas simulações com o *framework* SD6WSN. A partir dos experimentos realizados foi possível determinar a equivalência do desempenho do *framework* proposto com o roteamento tradicional RPL para tráfegos entre os nós e o 6LBR, e uma latência 30,87 % menor para tráfegos do tipo *peer-to-peer,* entre os nós da mesma RSSF. Além do aumento do desempenho da RSSF para determinados cenários, a abordagem proposta possibilita a construção de aplicações que podem explorar especificidades das RSSF, tais como o controle de potência, sobreposição de redes virtuais, priorização de pacotes e balanceamento de carga.

Palavras-Chave: RSSF, SDN, RPL, CoAP, 6LOWPAN

Abstract

This work proposes a framework that allows the application of the Software Defined Networking (SDN) paradigm in IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) Wireless Sensor Networks (WSN), consisting of an architecture and an SDN routing control protocol. In the proposed architecture, the routing decision of the data transmitted by the sensing applications residing at the nodes (data plane) is done by the network applications, which determine the most suitable paths for each type of flow within the WSN according to its specificities. The applications use the WSN unified view maintained by a controller, which gets the information from the nodes of the WSN. Routing decisions are materialized in table entries of flows installed on the nodes through message exchanges between the controller and the nodes. The exchange of messages between the controller and the nodes to maintain the unified view of the network and to maintain the entries in the flow tables (control plane) is performed by the Software Defined 6LoWPAN Wireless Sensor Network Protocol (SD6WSNP), proposed in this work. The SD6WSNP is based on the application layer protocol called Constrained Application Protocol (CoAP) which uses the IP/UDP stack with Routing Protocol for Low Power and Lossy Networks (RPL) as the transport protocol. Packet routing in the data plane is performed by agents installed on the wireless devices, based on the information contained in the flow tables. Through SD6WSNP messages, agents receive routing information from data flows, send information about the node connectivity (existing neighbors and link quality), physical parameters of the radio signal, and characteristics of the received data packets for which there is no match in the flow table. The choice of paths within the WSN is performed by applications connected to the controller, which, based on the information collected and the constraints of each node, establishes the best option for each data flow. For the validation of the proposal, the programs for the nodes were developed in the "C" language on the Contiki platform and the controller application was written in the Node.js language, in the Linux environment. The SD6WSN framework was validated by simulation in two typical

application scenarios for WSNs, in an Advanced Metering Infrastructure (AMI) and in a scenario with grid-type topology. The network simulations were performed in the Contiki/COOJA environment, on which the performance of an application that calculates the minimum paths between the 6LBR and the WSN nodes was compared using two performance metrics, packet receive ratio and latency. In the "grid topology" scenario, the latency was compared between node pairs belonging to the same RSSF for routes defined by an SD6WSN application in relation to those defined by RPL. It was also evaluated the overhead introduced by the control plane SD6WSN, measuring the proportion between the number of packets transmitted in the simulations with the RPL in comparison with the packets transmitted in the simulations with the SD6WSN framework. Based on the experiments performed, it was possible to determine the equivalence of the performance of the proposed framework with the traditional RPL routing for traffic between nodes and 6LBR, and a 30.87 % lower latency for peer-to-peer traffic between nodes of the same WSN. In addition to the increase in WSN performance for certain scenarios, the proposed approach allows the construction of applications that can exploit WSN specifics, such as power control, virtual network overlap, packet prioritization and load balancing.

Keywords: WSN, SDN, RPL, CoAP, 6LOWPAN

Capítulo 1

Introdução

As RSSFs (Redes de Sensores Sem Fio), incluindo as LLNs (*Low Power and Lossy Networks*), têm sido cada vez mais utilizadas para a aquisição de dados e controle de dispositivos remotos por suas características de simplicidade e baixo consumo de energia, o que as indica em casos onde a complexidade ou o custo de um sistema de comunicação de dados convencional dificultaria ou mesmo impediria a sua implementação [CUL2004].

A possibilidade de acesso direto e bidirecional aos sensores que formam a RSSF, dentro do conceito IoT (*Internet of Things*), aumenta consideravelmente as possíveis aplicações deste tipo de rede, porém traz alguns desafios no que tange à complexidade de interligação destas à Internet.

A interação entre os nós das RSSF com múltiplos saltos permite que cada nó atue no papel de roteador das informações transmitidas pelos nós mais distantes, criando assim uma rede que necessita menor potência de transmissão e, portanto, menor consumo de energia do que uma rede sem fio com topologia ponto-multiponto. A formação deste tipo de rede, a composição das adjacências e a melhor forma de encaminhamento dos dados tem sido tema de diversos trabalhos, como os citados por Palattella *et. al.* [PAL2013] para a criação de padrões destinados a otimizar a comunicação entre os nós da rede.

Uma premissa básica do conceito IoT é a comunicação fim-a-fim entre os nós e a Internet. A comunicação fim-a-fim permite que dispositivos sejam acessados diretamente em tempo real, sem a necessidade de dispositivos denominados *gateways*, que teriam o papel de realizar a conversão entre os protocolos utilizados nas RSSF e os protocolos empregados na Internet. Para evitar a conversão de protocolos citada, foi desenvolvido para as RSSF um padrão denominado 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*)

[RFC4919], que possibilita um endereçamento dos nós de uma RSSF com endereços IPv6, o mesmo protocolo que está sendo adotado na Internet e permite o acesso aos nós de qualquer lugar do mundo.

Uma área que vem se distinguindo pela necessidade de comunicação de baixo custo e de baixas taxas de transferência é a AMI (*Advanced Metering Infrastructure*), onde tanto as leituras de consumo de energia feitas por milhares de medidores devem ser encaminhadas para coletores centrais quanto alguns comandos, como corte e religação de consumidores, podem ser efetuados remotamente. Além da utilização em AMI, as RSSFs 6LoWPAN vêm sido adotadas em um contexto mais amplo como NANs (*Neighbor Area Network*) em *Smart Grids* [HAR2017] de empresas do setor elétrico por permitir a comunicação bidirecional para um controle mais efetivo da rede de distribuição de energia elétrica. Este controle contempla além de medição, o controle de cargas dos consumidores e o controle de geração fotovoltaica distribuída, entre outras aplicações.

Na área de redes de computadores, um novo paradigma que vem evoluindo nos últimos anos é o SDN (*Software Defined Networking*) [MCK2008], onde os protocolos tradicionais de roteamento são substituídos por uma nova abordagem centralizada, tendo o controle da rede transferida para elementos centrais, denominados "controladores", que determinam os caminhos que os pacotes tomam dentro da rede em função do fluxo a que pertencem e informam as decisões através de um plano de controle entre os elementos de rede e os controladores.

A aplicação do conceito SDN em RSSFs é um tema que está em desenvolvimento por suas especificidades, e que este trabalho procurou abordar.

1.1. Caracterização do Problema

Atualmente as decisões de roteamento entre os nós de RSSF que adotam o padrão 6LoWPAN ficam a cargo de protocolos de roteamento dinâmicos, principalmente o RPL (*Routing Protocol for Low Power and Lossy Networks*) [RFC6550], que são executados nos próprios nós, que, através de informações obtidas de seus vizinhos, decidem qual a melhor forma de encaminhamento dos dados. Esta abordagem tradicional possui diversos aspectos positivos, como a autonomia dos nós em escolherem os melhores caminhos, porém algumas restrições podem ser apontadas [CLA2011]:

- Complexidade computacional em elementos com baixo poder de processamento;
- Visão limitada da rede como um todo por parte de cada nó;

- Não existência de uma visão unificada da rede.

A utilização de um controle central para a tomada de decisões de encaminhamento de pacotes dentro de uma rede é uma alternativa aos protocolos de roteamento descentralizados e vem ao encontro com a abordagem SDN, onde o plano de controle da rede é desacoplado do plano de encaminhamento de dados. Adotado inicialmente em redes cabeadas, onde a conexão entre os dispositivos é realizada por cabos metálicos ou de fibra óptica, o conceito de Redes Definidas por *Software* vem se expandindo para outros tipos de redes, como as LANs (*Local Area Networks*) móveis celulares, Wi-Fi [YAP2011], WANs (*Wide Area Networks*) e inclusive as RSSF como as SDWSNs (*Software Defined Wireless Sensor Networks*), que é tema deste trabalho.

Cada tipo de rede possui suas características específicas, com soluções sendo desenvolvidas especialmente para endereçar estas diferenças. No caso das RSSF, uma série de medidas pode ser adotada para a sua operação dentro do conceito SDN, levando em conta as limitações inerentes a elas, tais como: baixo consumo de energia, tolerância à perda de pacotes, pouco poder de processamento e memória limitada dos nós.

Neste trabalho foi desenvolvido um *framework* denominado SD6WSN *(Software Defined 6LoWPAN Wireless Sensor Network*), formado por uma arquitetura e um protocolo, que se propõe a controlar o comportamento do tráfego de dados de RSSF seguindo a abordagem SDN.

A arquitetura proposta possui um controlador que se comunica com os nós através de um plano de controle, que continua utilizando o RPL como protocolo de roteamento para a manutenção do caminho entre os nós e o controlador. Através de mensagens do protocolo SD6WSNP (*Software Defined 6LowPAN Wireless Sensor Network Protocol*) proposto neste trabalho, o controlador envia aos nós as regras específicas para cada nó, ditadas por aplicações SD6WSN, que determinam qual o encaminhamento que cada tipo de pacote deverá ter na rede.

A manutenção do RPL permite a coexistência de nós SD6WSN com nós convencionais, permitindo uma migração gradativa de redes pré-existentes, sem que haja paralisação dos serviços que estejam em operação.

O controle centralizado dos caminhos dentro da RSSF permite a implementação de funcionalidades complexas para serem realizadas em redes tradicionais, tais como o balanceamento de tráfego, determinação de caminhos preferencias para tráfegos prioritários e determinação de políticas de segurança para controle de acesso no ingresso de pacotes. Outra

aplicação seria a segregação de nós que possuam limitações de consumo de energia ou com más condições ambientais na função de roteamento de pacotes para os nós vizinhos, poupando assim seus recursos escassos.

O principal benefício almejado pelo *framework* proposto é o controle centralizado do tráfego dos pacotes de dados dentro da rede sem fio, proporcionando uma melhor distribuição de tráfego entre os nós roteadores, priorização de tipos de dados específicos por determinados caminhos, melhor suporte às comunicações bidirecionais e entre os sensores e formação de redes virtuais, isolando o tráfego das aplicações.

Por meio dessa arquitetura, outros algoritmos diferentes do empregado pelo RPL podem ser adotados para o controle de tráfego entre os nós, por meio de uma nova aplicação para o controlador, trazendo uma flexibilidade hoje inexistente em redes 6LoWPAN. Outro benefício importante é a ampliação da capacidade de planejamento e gestão das RSSFs, com a inclusão de mecanismos que permitam o controle da interface de rádio, como a alteração da potência de transmissão dos determinados nós, diminuindo assim a interferência entre eles e, consequentemente, aumentando a taxa de recebimento de pacotes.

1.2. Objetivo geral

Propor, implantar e avaliar um *framework* para a adoção do paradigma SDN em redes sem fio com múltiplos saltos que adotem o padrão 6LoWPAN.

1.3. Objetivos específicos

Os objetivos específicos do presente trabalho são:

- Investigar as características das redes 6LoWPAN/RPL;
- Propor e especificar uma arquitetura baseada no paradigma SDN para as RSSFs;
- Especificar um protocolo *Southbound*, para a comunicação entre os nós e o controlador SDN;
- Especificar os processos envolvidos na comunicação entre os componentes da solução;
- Validar da arquitetura em ambiente simulado para um cenário de redes de medição de energia (AMI).

.

1.4. Motivação

A arquitetura SDN foi concebida para trazer à área de redes de computadores o avanço experimentado por outras áreas da computação que ocorreram nos últimos vinte anos [MCK2008]. Os protocolos de roteamento tradicionais dependem de decisões tomadas em conjunto pelos dispositivos, que nem sempre é a mais adequada, devido a visão limitada destes dispositivos em relação à rede em si e aos dados que trafegam por ela.

Novas funcionalidades, como balanceamento de tráfego, utilização eficiente de enlaces, prevenção contra *loops*, QoS (*Quality of Service)* e a separação lógica de tráfego utilizam técnicas complexas em redes convencionais, e que em uma abordagem SDN são mais simples e flexíveis, por terem uma inteligência centralizada, com a visão sistêmica da rede. A aplicação ou a atualização de regras de encaminhamento de pacotes é muito facilitada quando há uma visão unificada da rede.

Ter um controle centralizado não significa que toda a rede seja centralizada, pois as tabelas contendo as informações sobre os fluxos e as ações a serem tomadas ficam residentes nos dispositivos distribuídos, e somente novas entradas e modificações nas ações é que tem a participação do controle centralizado. Para o aumento da resiliência, os controladores podem ser replicados em diversas instâncias, que compartilham as mesmas informações.

Muitos dos problemas encontrados em redes cabeadas também estão presentes nas RSSFs, e sendo assim, a abordagem SDN é uma alternativa a ser estudada para a aplicação neste tipo de rede, respeitando-se suas idiossincrasias. Muitas das características únicas das RSSFs, como as limitações de *hardware* dos dispositivos, restrições de utilização de banda e a característica de ser susceptível a perdas de pacotes dificultam a adoção de um protocolo tradicional SDN como o *OpenFlow*, que foi projetado para redes cabeadas.

Desta forma, procurou-se desenvolver um *framework*, composto de uma arquitetura e um protocolo de mensagens de controle para as RSSFs, que contemplasse as suas características e limitações únicas.

1.5. Contribuições

Este trabalho apresenta um *framework* que permite a implantação do paradigma SDN em RSSF baseadas em 6LoWPAN, com a manutenção da compatibilidade com rede e dispositivos que porventura já existam.

O controle da rede proporcionado pelo *framework* proposto possibilita a execução de planejamento de tráfego além de permitir outras ações, como o controle de potência de transmissão de acordo com o destino de cada pacote. Este conjunto de recursos permitem o desenvolvimento de aplicações centralizadas para a otimização de RSSFs, o que seria muito custoso com a utilização dos protocolos atualmente empregados.

1.6. Estrutura do documento

No Capítulo 2 são apresentados os conceitos referentes às tecnologias empregadas neste trabalho que abrangem as camadas física e de enlace definidas pelo padrão IEEE 802.15.4, o padrão 6LoWPAN e o protocolo de roteamento RPL, e o protocolo de camada de aplicação CoAP (*Constrained Application Protocol*). Também é apresentado o conceito de SDN, com um detalhamento do protocolo *OpenFlow*.

No Capítulo 3 são comentados alguns trabalhos relacionados dentro da área de RSSF, principalmente os que abordam o protocolo RPL e o roteamento tradicional para redes 6LoWPAN além de trabalhos que versam sobre o tema SDWSN.

O Capítulo 4 apresenta os componentes da arquitetura SD6WSN, as mensagens do protocolo SD6WSNP, os processos de troca de mensagens para a coordenação da rede e são descritas algumas aplicações que foram idealizadas para esta arquitetura.

No Capítulo 5 é descrita a construção do ambiente computacional para o desenvolvimento, simulação e avaliação de redes baseadas no *Framework* SD6WSN na plataforma Contiki, tanto em dispositivos físicos como em simulação.

No Capítulo 6 é realizada a avaliação do desempenho do *Framework* SD6WSN em um ambiente simulado Contiki/COOJA.

As conclusões e os trabalhos futuros são apresentados em “Conclusão”.

Capítulo 2

Fundamentação Teórica

Neste capítulo são abordados os temas que foram estudados para a elaboração deste trabalho, iniciando-se com as redes de sensores sem fio 6LoWPAN que utilizam o padrão IEEE 802.15.4 em suas camadas físicas e de enlace e que tem o protocolo RPL com protocolo de roteamento. Em seguida é apresentado o protocolo de camada de aplicação CoAP, utilizado na transmissão de mensagens sobre RSSFs e, por fim, é feita uma descrição da tecnologia SDN e do protocolo *OpenFlow*, que serviu de base para a formatação do protocolo apresentado no capítulo 3.

2.1. Redes de sensores sem fio padrão IEEE 802.15.4

Os requerimentos das LLNs para diversas aplicações foram especificados nas RFCs (*Request for Comments*) [RFC5867] "*Building Automation Routing Requirements in Low-Power and Lossy Networks*", que trata das LLNs aplicadas em automação em prédios, [RFC5826] "*Home Automation Routing Requirements in Low-Power and Lossy Networks*" com foco nas LLNs de automação residencial, [RFC5673] "*Industrial Routing Requirements in Low-Power and Lossy Networks*" para LLNs industriais e [RFC5548] "*Routing Requirements for Urban Low-Power and Lossy Networks*" para as LLNs em ambientes urbanos.

O padrão IEEE 802.15.4 foi concebido para criar uma especificação para as camadas físicas e de controle de acesso ao meio (MAC) para redes sem fio de âmbito pessoal (WPAN), com características de baixa velocidade e baixo consumo de energia. A especificação original de 2003 [IEEE2003] determinou uma banda de até 250 kbps para comunicações na frequência de 2,4 GHz em todo o mundo, além das frequências de 868 MHz e 915 MHz (Figura 2.1) para determinadas regiões, porém com taxas mais baixas (Tabela 2.1). As propriedades de

propagação em radiofrequência nos diferentes espectros delimitam a aplicabilidade dos rádios, indicando o uso de frequências abaixo de 1 GHz para ambientes externos, devido ao seu maior alcance.

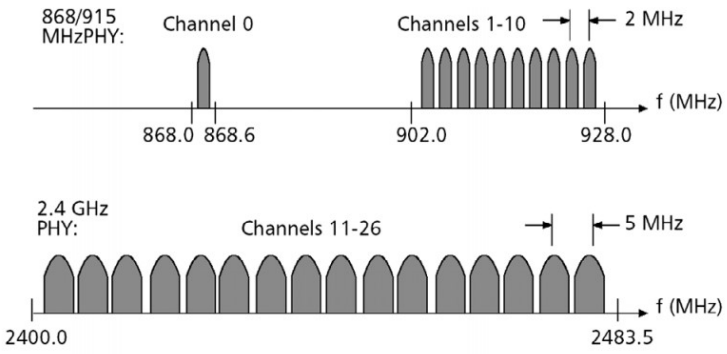

Figura 2.1 - Distribuição dos canais IEEE 802.15.4 [IEEE2003]

Tabela 2.1 - Características do padrão IEEE 802.15.4

Bandas	Banda ISM 2.4 GHz	915 MHz Estados Unidos	868 MHz Europa
Modulação	O-QPSK 250 kbps	BPSK 40 kbps	BPSK 20 kbps
Potência Máxima	20 dBm	> 10 dBm	30 dBm
Número dos canais	11 a 26	1 a 10	0
MTU	127 bytes por quadro (incluindo cabeçalhos)		
Payload	72 a 116 bytes por quadro		
Compartilhamento do canal	CSMA/CA		
Segurança na camada de enlace	Encriptação AES de 128 bits		
Endereçamento	Endereços de 64 bits (longo) ou 16 bits (curto) com capacidades *unicast* ou *broadcast*		

2.1.1. Camada física 802.15.4

O padrão mais adotado em tecnologia de rádio de baixa potência é o IEEE 802.15.4 que se tornou o padrão de fato para WPANs (*Wireless Personal Area Networks*). Este padrão define a camada física (PHY), que envolve as modulações e frequências utilizadas e a camada de enlace (MAC), de endereçamento entre os nós. A primeira revisão foi publicada em 2003, com

revisões subsequentes em 2006 e 2011. Os grupos de trabalho envolvidos no aprimoramento deste padrão são identificados por letras, como, por exemplo, IEEE 802.15.4g.

A especificação da camada PHY do IEEE 802.15.4 procurou equilibrar vários fatores, como baixo custo, baixo consumo de energia, alcance limitado à vizinhança e taxa de transferência de dados adequada para o estabelecimento de redes de vizinhança ou para utilização em edifícios.

Embora o padrão defina várias faixas de frequência que podem ser utilizadas (Tabela 2.1), a mais utilizada é a situada na faixa não licenciada de 2,4 GHz.

O padrão IEEE 802.15.4 requer que os rádios tenham uma sensibilidade mínima de recepção de -85 dBm, mas na prática os circuitos integrados utilizados conseguem chegar a sensibilidades entre -95 dBm e -100 dBm. A camada física para a faixa de 2,4 GHz utiliza a modulação O-QPSK (*Offset-Quadrature Phase-Shift Keying*), com 2 Mbps de taxa de dados física. Internamente ao rádio, cada grupo de quatro bits de dados enviados para a transmissão é codificado em 32 chips (também denominados "bits físicos") seguindo uma tabela de pesquisa. Esta técnica, denominada DSSS (*Direct Sequence Spread Spectrum*), traz maior confiabilidade na comunicação, mas faz com que a taxa efetiva seja de apenas 250 kbps em um canal de 2 Mbps, pois internamente 8 chips a mais são enviados [PAL2013].

Para a faixa de 2,4 GHz, o IEEE 802.15.4 define 16 canais ortogonais de 2 MHz distantes entre si em 5 MHz, localizados entre 2,405 GHz e 2,480 GHz. Esta faixa de frequência, denominada ISM (*Industrial, Scientific and Medical*), é de uso livre, porém é compartilhada com outras tecnologias de comunicação sem fio, como o IEEE 802.11, destinada às redes Wi-Fi. Somente os canais 15, 20, 25 e 26 estão localizados em frequências não utilizadas pelos canais Wi-Fi 1,6 e 11, mais comumente empregados nestas redes, por serem ortogonais entre si.

Os rádios padrão IEEE 802.15.4 podem enviar e receber dados em qualquer um dos canais, com um tempo de comutação entre canais de no máximo 192 μs. Quando um rádio envia um pacote, ele inicia o processo transmitindo um preâmbulo físico de 128 μs para permitir ao receptor sincronizar com seu sinal. Em seguida o transmissor envia um SFD (*Start of Frame Delimiter*) para indicar o início da transmissão dos dados. O primeiro byte dos dados indica o tamanho (em bytes) do pacote, que tem um tamanho máximo de 127 bytes (128 bytes incluindo-se o byte de tamanho). A Figura 2.2 apresenta o formato de camada física de um quadro IEEE 802.15.4.

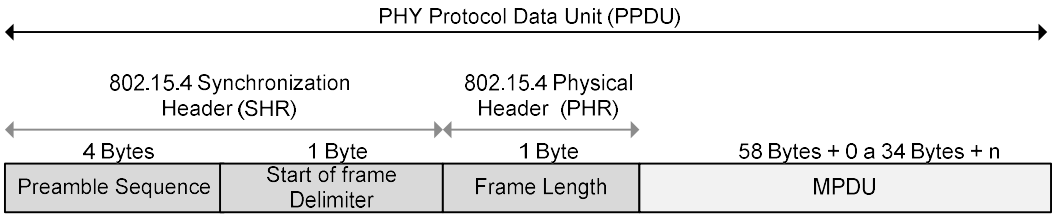

Figura 2.2 – Formato de um quadro IEEE 802.15.4 (camada física) [IEEE2003]

Um rádio fica permanentemente na condição de recepção até receber o preâmbulo físico para que ele possa se sincronizar. Após o sincronismo, ele aguarda o SFD e o byte de comprimento, e então preenche o *buffer* de recepção com os demais bytes indicados no byte de comprimento.

2.1.2. Camada MAC IEEE 802.15.4-2003

O padrão IEEE 802.15.4 também define a camada de acesso ao meio (MAC), que permite a interação direta com o rádio. No padrão, é especificado o formato do cabeçalho MAC (campos dos endereços de origem e destino) e como os nós sensores, também conhecidos como *motes*, se comunicam. A Figura 2.3 apresenta o formato de camada MAC de um quadro IEEE 802.15.4 [IEEE2003].

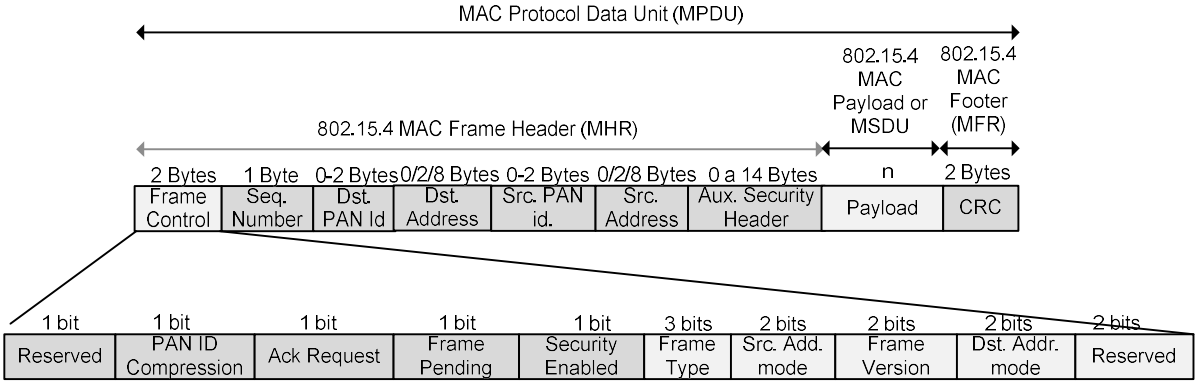

Figura 2.3 – Formato de um quadro IEEE 802.15.4 (camada MAC) [IEEE2003]

Algumas características da camada MAC são:

- Adequação a tempo-real pela reserva garantida de fatias de tempo;
- Prevenção de colisão através do CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*);
- Suporte integrado a comunicações seguras (encriptação 128-bit AES);

- Possibilidade (teórica) de criação de topologias “estrela” e “malha”;
- Suporte para baixas latências e endereçamento dinâmico dos dispositivos.

O protocolo MAC de IEEE 802.15.4-2003 é adequado para topologias estrela simples e, embora ele permita a formação de estrelas estendidas de múltiplos saltos, tais topologias sofrem do fato de que muitos dos nós da rede precisam manter o rádio sempre ligado (ciclo de trabalho de 100 %), e que toda a comunicação na rede acontece num único canal. Para endereçar as limitações relativas ao ciclo de trabalho RDC (*Radio Duty Cycle*) e criar um padrão que permitisse a utilização de múltiplos canais divididos em *timeslots*, foi criado em 2008 um grupo de trabalho que originou o IEEE 802.15.4e [IEEE2012].

2.2. Dispositivos de recursos limitados

Os dispositivos com transceptores IEEE 802.15.4, também denominados ”*motes*”, são compostos por um circuito integrado com as funções de receptor, e amplificador de potência (PA) e um microcontrolador, com um código que implementa as funções das camadas MAC do protocolo. Alguns fabricantes, como a Texas Instruments, agregam o rádio com o microcontrolador em um único circuito integrado, denominado SoC (*System on a Chip*).
As principais características destes dispositivos são baixo custo e baixo consumo de energia, associados a baixas capacidades de processamento, memória Flash e memória RAM. Por possuírem essas características limitantes, são classificados como “dispositivos de recursos limitados”.

Em 2010 foi criado um grupo de trabalho do IETF denominado CoRE (*Constrained RESTful Environments*) que teve o objetivo de criar um arcabouço para aplicações em redes IP utilizando dispositivos de recursos limitados. Em [RFC7228], os tipos de *motes* foram classificados em função de seus recursos em três classes: 0, 1 e 2 (Tabela 2.2). Apesar da “Lei de Moore” determinar um aumento da capacidade computacional com o passar do tempo, esta RFC assume que ela não será tão efetiva para dispositivos embarcados, pois o foco para estes dispositivos será a diminuição de custo e consumo de energia em detrimento ao número de transistores e capacidade de processamento.

Tabela 2.2 – Classes de dispositivos de recursos limitados [RFC7228]

Nome	Tamanho da área de dados (RAM)	Tamanho da área de código (Flash)
Classe 0, C0	<< 10 kbytes	<< 100 kbytes
Classe 1, C1	~ 10 kbytes	~ 100 kbytes
Classe 2, C2	~ 50 kbytes	~250 kbytes

Os dispositivos da classe 0 não possuem recursos suficientes para a execução de uma pilha IP de uma forma segura, necessitando de *gateways* dos protocolos proprietários utilizados nesses dispositivos para que seja efetuada a conexão com redes IP/IPv6, como a Internet. Para que seja possível a implementação completa de uma pilha IPv6 com recursos de segurança como o DTLS (*Datagram Transport Layer Security*), há a necessidade da adoção de um dispositivo de classe 1 ou superior.

O SoC modelo CC2538 [TI2012], que possui um processador com arquitetura ARM Cortex-M, transceptor IEEE 802.15.4 em 2.4 GHz, 512KiB de memória Flash e 32 KiB de RAM, portanto, de classe 1, permite a construção de *motes* equipados com *software* totalmente compatíveis com o trabalho do grupo de trabalho CoRE. A Figura 2.4 apresenta um *mote* construído com este SoC.

Figura 2.4 – *Mote* com o SoC T.I CC2538

Neste trabalho, onde é necessária uma pilha IPv6 completa provida pelo Sistema operacional Contiki e suporte ao protocolo de aplicação CoAP, foram empregados *motes* de classe 1.

2.3. Redes 6LoWPAN

Para se obter uma conectividade fim-a-fim via Internet, os pacotes IP normalmente transitam por redes interconectadas por diversas tecnologias de enlace. Para se mapear os serviços requeridos pela camada IP nos serviços oferecidos pela camada inferior (isto é, a camada de enlace), algumas vezes é necessário introduzir uma subcamada intermediária, também denominada "camada de adaptação" [SHE2010], que leve em conta as especificidades presentes nestas camadas.

As RSSFs de baixa potência são caracterizadas por: pacotes de tamanho reduzido, suporte para endereços com diferentes comprimentos, baixa largura de banda, topologias estrela e malha, dispositivos de baixo custo alimentados por bateria, grande número de dispositivos, posições dos nós desconhecidas, baixa confiabilidade e longos períodos ociosos, principalmente quando as interfaces de comunicação estão desligadas para economia de energia [RFC4919].

Outras características abrangem a autoconfiguração de endereços IPv6 [RFC2464]; o cumprimento da recomendação para o suporte ao *broadcast* na transmissão de sub-rede da camada de enlace em redes compartilhadas [RFC3819]; a redução de roteamento e gerenciamento de *overhead*; a adoção de protocolos de aplicação leves (ou novas técnicas de codificação de dados); mecanismos de segurança (ou seja, que garantam confidencialidade e integridade dos dados); inicialização dos dispositivos e criação e gerenciamento de chaves.

Dadas as características acima mencionadas, pode-se perceber que a adoção de IPv6 no topo de uma RSSF de baixa energia não é simples e coloca fortes exigências na otimização desta camada de adaptação. Por exemplo, devido ao padrão IPv6 possuir tamanho mínimo de MTU (ou seja, 1280 bytes), um pacote não fragmentado IPv6 seria grande demais para caber em um quadro IEEE 802.15.4. Além disso, o *overhead* devido aos 40 bytes do cabeçalho longo do IPv6 desperdiçaria a escassa largura de banda disponível na camada PHY.

Mesmo nas RSSFs onde o consumo de energia não é um problema, como no caso das redes AMI onde os nós são alimentados, o desperdício causado pelo *overhead* do IPv6 deve ser evitado, já que a capacidade da rede e as velocidades de transmissão são limitadas.

A Figura 2.5 mostra a composição de um pacote IPv6 sobre o IEEE 802.15.4.

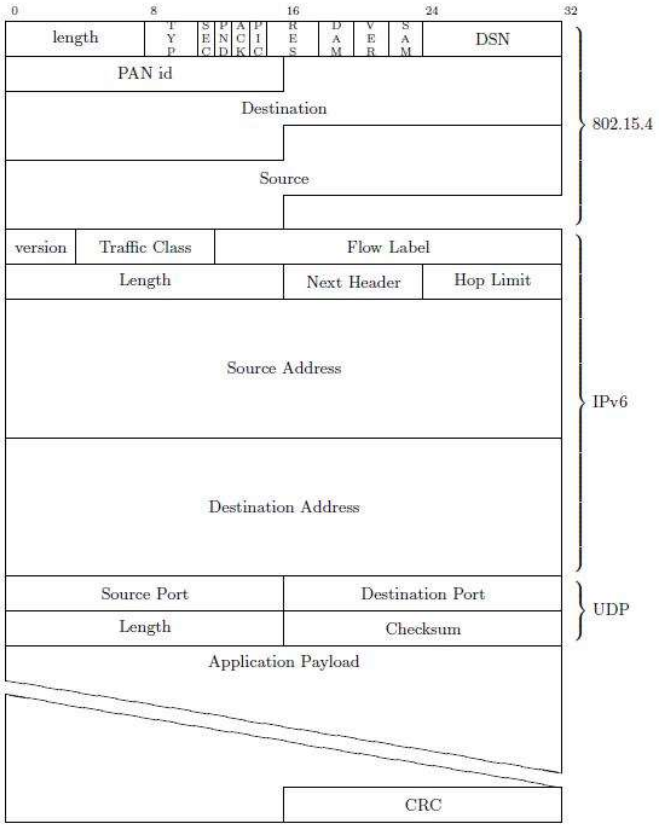

Figura 2.5 - Cabeçalhos de um pacote IPv6 sobre IEEE 802.15.4 [SHE2010]

Através de estudos da interação entre o padrão IEEE 802.15.4 e os protocolos IPv6 e UDP (*User Datagram Protocol*), o grupo de trabalho 6LoWPAN definiu uma camada de adaptação eficaz, removendo campos que são redundantes nos cabeçalhos e reduzindo assim o tamanho dos pacotes que estão sendo transmitidos por radiofrequência. O resultado do grupo de trabalho deu origem ao padrão 6LoWPAN, documentado em [RFC4944] e [RFC6282].

Na Figura 2.6 pode-se observar as diferenças entre as pilhas de protocolos do protocolo IP e do 6LoWPAN, onde se percebe a introdução de uma camada de adaptação denominada LoWPAN entre a camada de enlace (MAC) e a de rede IPv6.

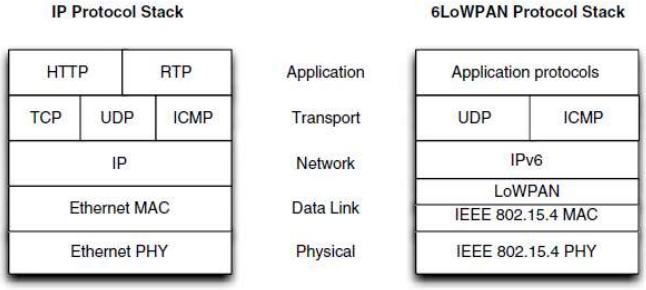

Figura 2.6 - Comparação entre as pilhas de protocolos IP e 6LoWPAN [SHE2010]

Alianças como o "IP para *Smart Objects*"[1] fornecem uma indicação clara de que este tipo de estrutura de pacotes padrão vai se tornar onipresente nas redes de amanhã de dispositivos sem fio com recursos limitados.

2.3.1. Especificação 6LoWPAN

A especificação 6LoWPAN presente na [RFC4944] é o resultado do grupo de trabalho homônimo do IETF (*Internet Engineering Task Force*) para otimizar o uso do IPv6 em redes sem fio de baixa velocidade baseadas no padrão IEEE 802.15.4.

O 6LoWPAN remove alguns campos nos cabeçalhos IPv6 e UDP por estes possuírem valores conhecidos, ou por seus valores poderem ser inferidos a partir de campos no cabeçalho IEEE802.15.4. No cabeçalho IPv6, o campo é sempre "versão 6" para IPv6, os campos *Traffic Class* e o *Flow Label* nunca são usados e o campo de comprimento *lenght* é sempre igual ao comprimento do campo IEEE802.15.4 menos o comprimento do cabeçalho IPv6. Todos esses campos podem, portanto, ser removidos. O cabeçalho *next-header* tipicamente aponta para UDP ou TCP (*Transport Control Protocol*), e assim campo de 8 bits pode ser substituído por um campo de 2 bits, como parte do campo HC1 do cabeçalho 6LoWPAN. Finalmente, em [RFC2464] define-se como os endereços IPv6 de 128 bits podem ser recuperados a partir de endereços MAC de 64 bits, como os campos de origem e destino do IEEE 802.15.4. Isso permite a remoção dos campos de endereço de destino e de origem IPv6. No final, apenas o campo de limite de saltos (*Hop Limit*) tem de estar presente no cabeçalho 6LoWPAN e, do mesmo modo para o UDP, o comprimento pode ser calculado a partir do campo de comprimento do IEEE 802.15.4. Nos casos mais comuns de utilização das redes de sensores, apenas um número limitado de portas é utilizado, de modo que quatro bits são suficientes para descrevê-las, em vez de 8 bits. A Figura 2.7 mostra o mesmo pacote da Figura 2.5, porém agora com o cabeçalho comprimido pelo 6LoWPAN.

[1] http://www.ipso-alliance.org – Acesso em 10 de janeiro de 2018.

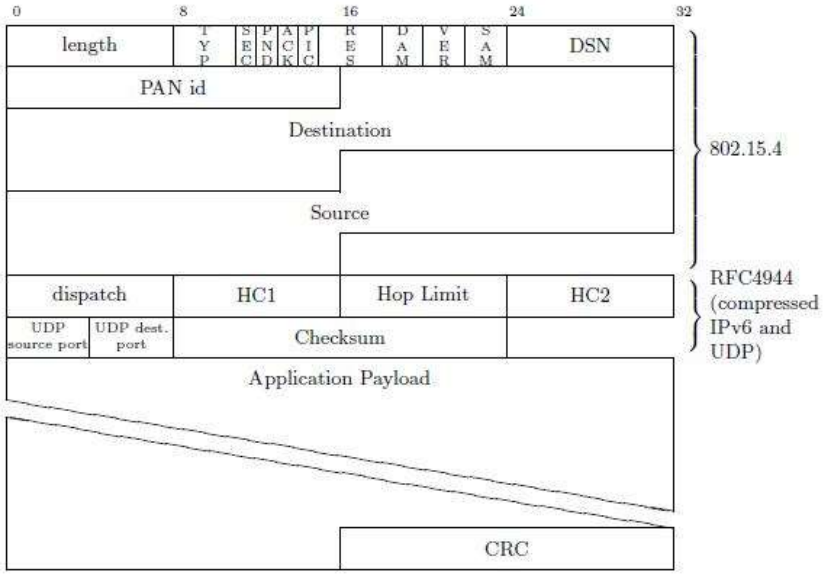

Figura 2.7 - Cabeçalhos de um pacote 6LoWPAN [SHE2010]

2.3.2. Compressão de cabeçalhos IPHC

Com o tempo, somente as definições da [RFC4944] mostraram-se insuficientes para usos mais práticos das redes 6LoWPAN. Ela se mostra eficaz na comunicação *unicast* link-local, onde os endereços IPv6 utilizam o prefixo *link-local* e um identificador de interface (IID), derivados diretamente dos endereços IEEE 802.15.4, onde ambos os endereços podem ser completamente omitidos. No entanto, os endereços de conexões *link-local* geralmente não são utilizados para o tráfego de dados de aplicações, de modo que a utilidade deste mecanismo de compressão é limitada. Para endereços IPv6 globais, a [RFC4944] requer que endereços de origem e destino IPv6 carreguem juntamente o prefixo do endereço.

Para tratar desta limitação, a técnica de compressão IPHC (*IP Header Compression*) definida na [RFC6282], comprime os endereços *IPv6 Unique Local, Global e multicast* baseada no estado compartilhado dentro de contextos. Além disso, ela introduz uma série de melhorias adicionais sobre o formato de compressão de cabeçalho definido na [RFC4944]. A IPHC também define o NHC (*Next-header Compression*), que é um formato de codificação para os *next-headers* arbitrários. A IPHC indica se o cabeçalho que se segue é codificado usando NHC. Se este for o caso, os bits seguintes ao cabeçalho IPv6 comprimido iniciam a codificação do NHC. Ele adiciona a capacidade de retirar o *checksum* UDP, o que economiza um par adicional de bytes. O NHC também ativa a compactação dos cabeçalhos de extensão IPv6.

Os formatos de quadros são apresentados no 6LoWPAN utilizando a compressão IPHC no lugar da [RFC4944]. Nota-se que o formato do cabeçalho IPHC é diferente quando o cabeçalho *next-header* é ICMPv6 (como, por exemplo, as mensagens DIO e DAO do protocolo RPL, que são casos em que o NHC não pode ser usado) ou UDP (dados de aplicação, caso em que é utilizado NHC). Nos dados de aplicação, por exemplo, o IPHC comprime até sete octetos do cabeçalho IPv6 (um octeto de expedição, um octeto IPHC, um octeto de *Hop Limit*, dois octetos de endereço de origem, e dois octetos de endereço de destino). O *Hop-Limit* não pode ser compactado porque ele precisa ser decrementado a cada salto e pode tomar qualquer valor.

2.3.3. Roteamento em redes 6LoWPAN

As RSSFs 6LoWPAN com múltiplos saltos, ou seja, que utilizam os próprios nós para o encaminhamento de pacotes originados ou destinados em nós mais distantes, necessitam que haja uma forma de se estabelecer este caminho que passa pelos nós intermediários. Os nós que pertencem a uma determinada RSSF possuem endereços IPv6 que estão na mesma sub-rede IPv6 e, portanto, a comunicação entre eles se daria por encaminhamento em camada de enlace, pela utilização do endereço MAC do nó de destino.

Diferentemente das redes cabeadas, onde os nós são diretamente alcançáveis entre eles, em RSSFs de múltiplos saltos não há o encaminhamento direto entre os nós por camada de enlace, mesmo os nós estando na mesma sub-rede. Por este motivo há a necessidade de se estabelecer mecanismos de encaminhamento de pacotes diferentes dos utilizados nas redes cabeadas.

O roteamento de múltiplos saltos entre os nós de uma rede 6LoWPAN pode ser realizado de duas formas:

a) Por meio de um protocolo *mesh* (como, por exemplo, o Zigbee), que realiza a função de estabelecer os melhores caminhos entre os nós.

   A Figura 2.8 ilustra a primeira forma de roteamento, denominada "*mesh-under*", onde o encaminhamento de pacotes ocorre diretamente na camada de enlace, abstraindo para as camadas superiores a complexidade da rede, como se os nós estivessem diretamente conectados e não em uma topologia de múltiplos saltos.

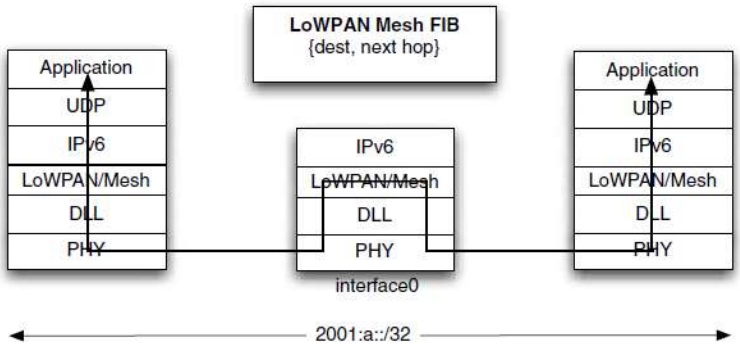

Figura 2.8 - Roteamento *"mesh-under"* [SHE2010]

b) A segunda forma consiste em deixar que um protocolo de roteamento como o RPL decida qual o *next-hop* dos pacotes através de um mecanismo de camada de transporte, que leva em conta os endereços IPv6 *link-local* dos nós para estabelecer o caminho. Esta forma de roteamento, que não necessita de um protocolo *mesh* de camada MAC, é denominada "*route over*", é ilustrada na Figura 2.9. Cada pacote é encaminhado para o endereço IPv6 de *next-hop* do nó intermediário (que atua como roteador) e em seguida é encaminhado para os próximos nós sucessivamente até que chegue à raiz da árvore (o destino).

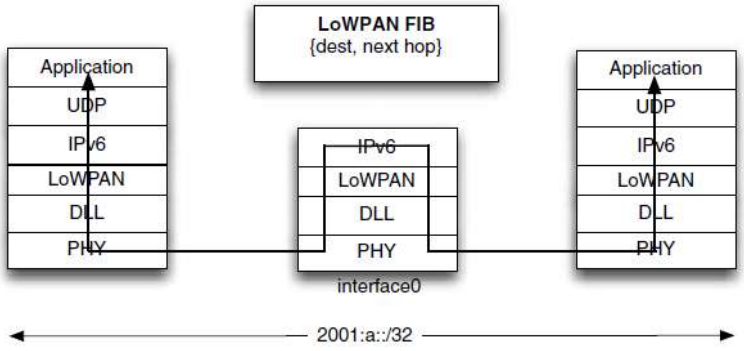

Figura 2.9 - Roteamento *"route-over"* [SHE2010]

Com a padronização do protocolo RPL pelo IETF, somente será abordada a formação de redes 6LoWPAN de múltiplos saltos pelo modelo *route-over*.

2.4. Roteamento RPL

O RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*) foi tema do grupo de trabalho "ROLL" do IETF, cujo estudo resultou na publicação da [RFC6550], definindo o

processo de formação de DODAGs (*Destination Oriented Directed Acyclic Graph*), que interconecta os nós de uma rede de sensores. Um DODAG é uma topologia sem ciclos, que se diferencia de uma topologia de árvore convencional pela característica de cada nó poder possuir pais alternativos. O DODAG foi escolhido por ser bem adequado às redes onde se pressupõe que o tráfego majoritário seja no sentido dos nós para a raiz (resultados de medidas, por exemplo), com um tráfego bem menor no sentido inverso e ainda onde o tráfego direto entre os nós seja desprezível.

De acordo com a arquitetura "em camadas" do protocolo IP, o RPL não se restringe à uma tecnologia de camada de enlace específica. Ele foi projetado para operar sobre diferentes tecnologias de enlace, desde as que apresentam restrições quanto ao consumo de energia ou sejam susceptíveis a perdas de pacotes, como no caso das redes sem fio, até as que operam em meios mais confiáveis, como PLC *(Power Line Communication*) ou Ethernet. Outra característica das RSSFs é que normalmente não possuem topologias pré-definidas como as impostas por redes cabeadas, e desta forma o RPL deve realizar a descoberta dos nós existentes ao alcance da rede.

2.4.1. Processo de construção da topologia de uma rede RPL

O processo de construção do DODAG [IPSO2011] tem início no LBR (*LoWPAN Border Router*), que é a raiz da rede definida previamente no projeto da rede. Para a comunicação entre a raiz e os nós, a [RFC6550] especificou diversas mensagens ICMPv6 [RFC4443], que têm a função de trocar informações relacionadas à montagem do grafo. As mensagens definidas são:

- **DIO** (DODAG *Information Object*);
- **DAO** (DODAG *Destination Advertisement Object*);
- **DIS** (DODAG *Information Solicitation*).

A formação do DODAG se inicia na raiz, e se espalha gradualmente pela LLN coberta pelo alcance das mensagens DIO, com a determinação dos conjuntos de pais para cada nó. As mensagens DIO enviadas pela raiz possuem um objeto de configuração que determina alguns atributos (tais como DODAG-ID, prefixo IPv6, *rank* da raiz) que serão utilizados por cada nó para estabelecer sua posição na rede, pelo cálculo de seu próprio *rank*, que é o valor que define qual é sua distância em relação à raiz do DODAG. Os atributos mais recentes vindos da raiz e

recebidos pelos roteadores são reenviados pelos roteadores em suas próprias mensagens DIO, propagando assim as possíveis alterações por toda a rede.

A frequência de envio das mensagens DIO é disciplinada pelo algoritmo *Trickle* [RFC6206] que, para proporcionar economia de recursos de energia e de banda, diminui a emissão de mensagens de controle RPL após a estabilização da rede.

Um exemplo de uma mensagem DIO é mostrado na Figura 2.10, onde se observa diversas informações, como a DODAG-ID, versão, prefixo IPv6 da sub-rede, entre outros parâmetros.

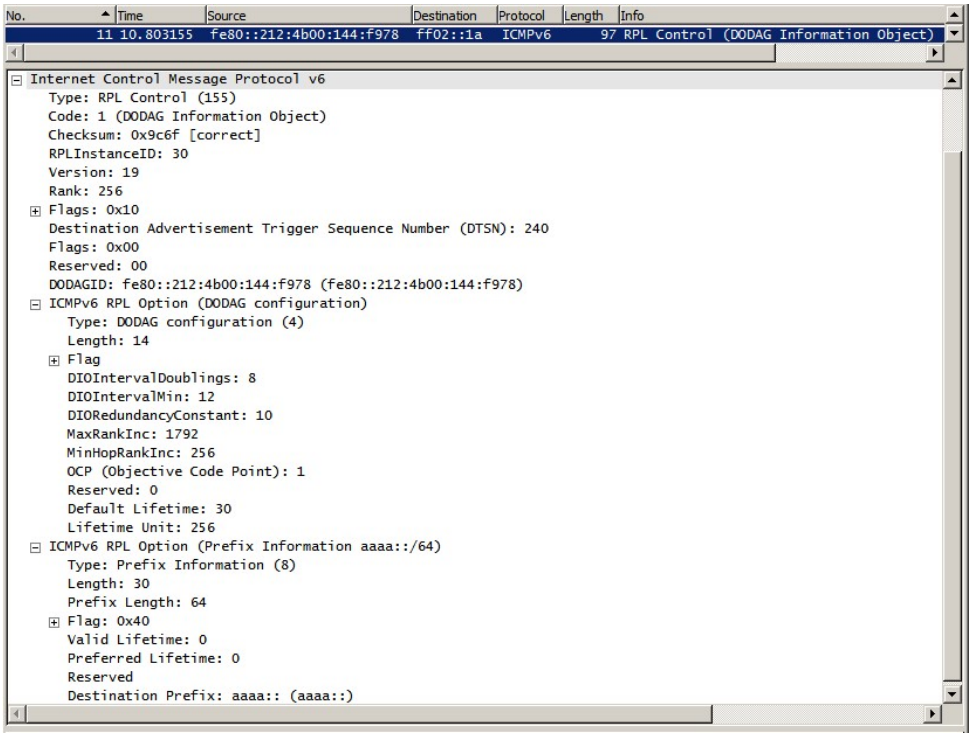

Figura 2.10 – Mensagem DIO

Após a junção de um nó ao grafo, este nó terá uma rota em direção à raiz do DODAG, passando pelo nó definido como seu “pai”.

Um nó pode estar previamente configurado como roteador ou como “folha”. Se ele for um nó “folha”, ele recebe mensagens DIO, mas não as gera. Porém, se ele for configurado como roteador, logo após a junção ao DODAG o nó começa a emitir suas próprias mensagens DIO. Assim que receberem estas mensagens, os nós vizinhos repetirão o processo inicial de seleção de nó pai, adição de rotas e emissão de mensagens DIO. Este efeito, denominado *rippling*, que consiste na recepção e transmissão de mensagens DIO a partir da raiz e se propagando até as

folhas, forma o grafo da rede RPL. Após esta montagem da rede, cada nó possuirá um pai preferencial e um conjunto de pais alternativos. Isto possibilita que os pacotes enviados por cada nó alcancem a raiz passando por nós intermediários, pois cada nó encaminha os pacotes recebidos para seu pai, formando assim um roteamento *hop-by-hop*.

Este modelo de encaminhamento onde cada nó tem conectividade ao nó raiz é denominado MP2P (multiponto-ponto). É também referenciado como roteamento *upward*, ou de direção de "subida", no DODAG. Os vários passos da construção do DODAG são representados na Figura 2.11, onde R é o nó raiz e os nós de 1 a 6 podem ser roteadores ou apenas nós folhas, conforme a topologia definida pelo cálculo do DODAG.

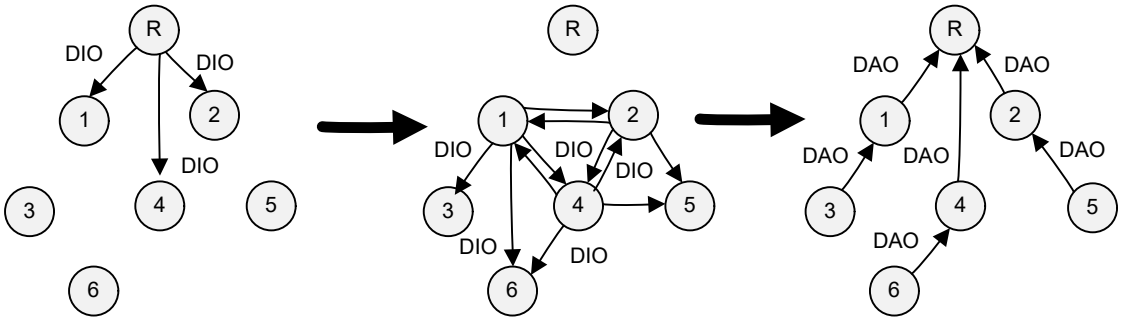

Figura 2.11 - Passos para a construção de um DODAG

Ao contrário das mensagens DIO, as mensagens DAO tem a função de prover a conectividade "*downward*", ou "de descida", dos pacotes originados na raiz ou fora da RSSF em direção dos nós intermediários ou folhas. Cada nó que ingressa no DODAG envia (via *unicast*) uma mensagem DAO para os "pais", com as suas informações de prefixo, tempo de vida e distância (entre outras). As mensagens DAO também podem ser solicitadas para a sub-DAG através de parâmetros específicos nas mensagens DIO. Um exemplo de uma mensagem DAO está representado na Figura 2.12, onde se observa o campo onde a informação do endereço global IPv6 do nó destino é informada aos nós antecessores, até chegar à raiz.

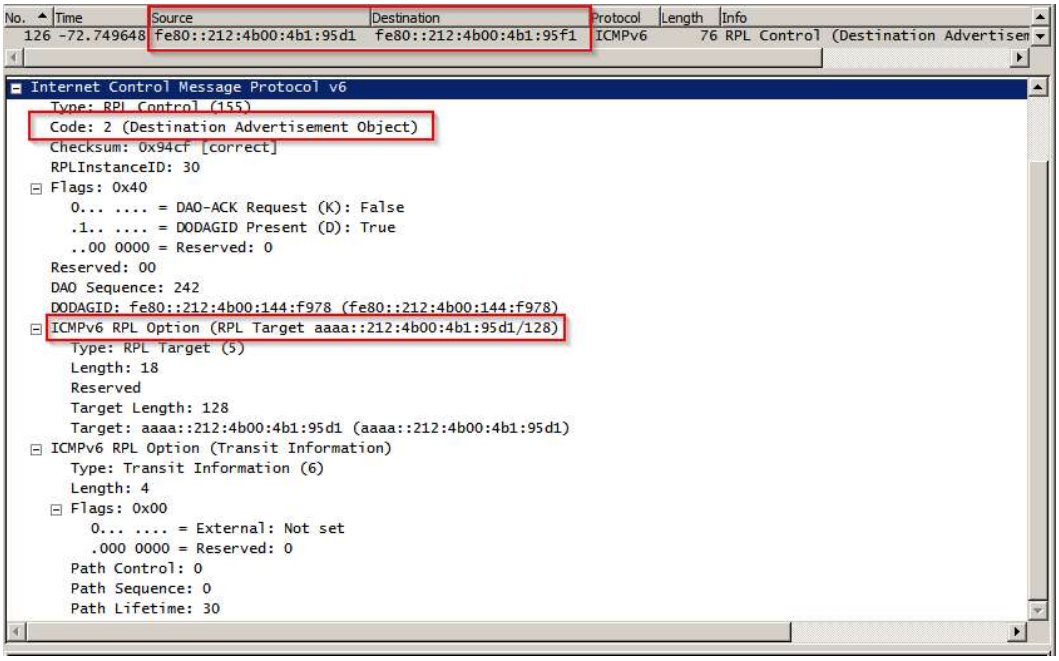

Figura 2.12 - Campo RPL Target dentro de uma mensagem DAO

Os roteadores em uma rede RPL podem operar em dois modos, definidos pelo *flag* MOP (*Mode of Operation*) no objeto de configuração informado pela raiz. No modo de operação "*non-storing-mode*" (Figura 2.13), o roteador RPL origina mensagens DAO direcionadas via *unicast* para a raiz informando um ou mais de seus pais. Quando a raiz recebe estes DAOs do roteador e de todos os roteadores envolvidos no caminho entre eles, ela utiliza roteamento pela origem para alcançar cada destino anunciado na LLN.

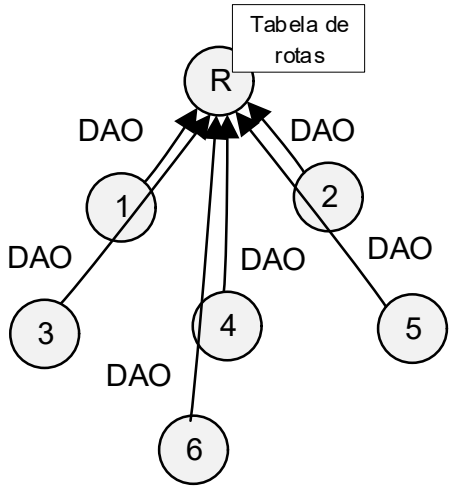

Figura 2.13 - Modo de operação "*non-storing-mode*"

No modo "*storing-mode*" (Figura 2.14), os roteadores no caminho entre o nó de origem da mensagem DAO e a raiz armazenam a rota para os prefixos e o *next-hop* anunciados na DAO e, em seguida, repassam a DAO para seu pai preferencial.

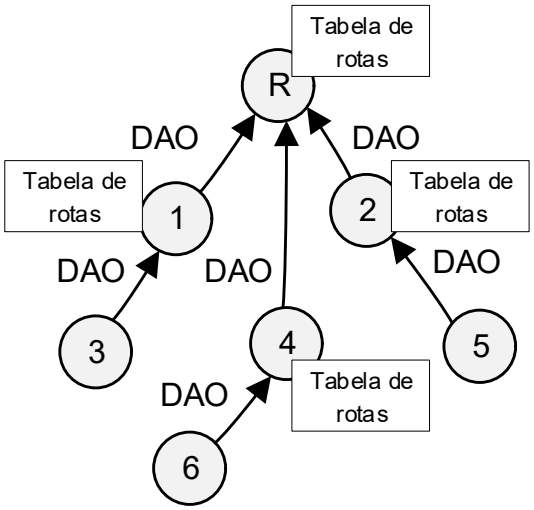

Figura 2.14 - Modo de operação "*storing-mode*"

A comunicação entre dois nós que não sejam raiz e pertencentes à mesma RSSF é possível, pois todos os nós possuem rotas *default* em direção à raiz para tráfego *upward*, onde então os pacotes podem ser encaminhados *downward* em direção a outros nós. A rota de *downward* depende do modo de operação (MOP) do DODAG. Se o modo for o *non-storing-mode*, uma informação de "roteamento pela origem" é adicionada para que o nó destino seja alcançado. Caso o modo de operação seja o *storing-mode*, as rotas *hop-by-hop* armazenadas nos nós roteadores já são suficientes para prover a conectividade ao nó final. Neste modo, caso os nós de origem e de destino possuam algum ancestral comum, o roteamento entre eles é realizado diretamente, sem a necessidade que os pacotes alcancem a raiz do DODAG.

A mensagem DIS é utilizada pelos nós para solicitar as informações do grafo (enviadas via mensagens DIO) de forma assíncrona, ou seja, fora dos tempos estabelecidos pelo algoritmo *Trickle*. São utilizadas principalmente por nós novos para anunciarem aos demais a sua presença.

Sendo um protocolo Vetor-Distância [RFC6550], o RPL restringe algumas possibilidades de um roteador mudar seu *rank*. O cálculo do *rank* de cada nó depende da função objetivo adotada na RSSF. A [RFC6552] define a função objetivo "OF0" que calcula o *rank* de

cada nó em função de número de saltos dele até a raiz, sem levar em conta as métricas de qualidade dos enlaces.

Em [RFC6551] especificou-se uma série de métricas de enlaces para serem usadas pelo o RPL para definir os *ranks* de cada nó, como o ETX (*Expected Transmission Count*), que é o número de retransmissões que foram necessárias até que um pacote seja recebido. Assim, quanto maior for este valor (podendo ser de 1 a 5), pior é a qualidade do enlace.

Em [RFC6719] especificou-se a função objetivo MRHOF (*Minimum Rank with Hysteresis Objective Function*) que utiliza o ETX como métrica e conta com um mecanismo para evitar o recálculo de *rank* por pequenas mudanças do ETX.

Cada roteador pode alterar seu *rank* para um valor menor (ou seja, mais perto da raiz) que o anteriormente divulgado caso encontre um ou mais pais com *rank*s menores que os atuais. Neste caso, ele deverá desconsiderar todo o conjunto de pais anterior. Já a alteração para um valor maior é restrita, pois pode desencadear problemas como "contagem ao infinito", onde todos os roteadores iniciam o processo de aumento de *rank*, desestabilizando a rede.

Uma mudança de *rank*s dos roteadores, tanto para um valor superior como para um inferior, pode ocorrer quando a raiz disparar um recálculo global do DODAG, que ocorre quando esta eleva o atributo "*DODAG version*" em suas mensagens DIO.

2.5. Camadas de transporte e de aplicação em redes 6LoWPAN

A camada de transporte gerencia as sessões de comunicação entre aplicativos em execução nos dispositivos finais. A camada de transporte permite que múltiplos aplicativos em cada dispositivo possam ter seu próprio canal de comunicação. Atualmente o TCP é o protocolo de transporte dominante na Internet. No entanto, o TCP é um protocolo orientado a conexão (inclui reordenação de pacotes) com grande *overhead* e, portanto, nem sempre adequado para dispositivos que exigem consumo de energia muito baixo. Para esses tipos de sistemas, o UDP, que é um protocolo sem conexão e que apresenta menor *overhead* pode ser uma opção mais adequada. Não há nada que impeça um sistema 6LoWPAN operar com TCP, mas a compressão de cabeçalho TCP não faz parte da [RFC6282]. Os exemplos de camadas de transporte seguras incluem a TLS (*Transport Layer Security*) sobre TCP e o DTLS, que é baseado em UDP, e que pode ser utilizado opcionalmente para o transporte seguro de mensagens CoAP.

Por fim, a camada de aplicação é responsável pela formatação de dados. Ela também garante que os dados são transportados em esquemas otimizados para a aplicação. A camada

de aplicação mais amplamente utilizada na Internet é o HTTP (*Hypertext Transfer Protocol*) sobre o TCP, porém o HTTP utiliza o XML (*Extensible Markup Language*), que é uma linguagem baseada em texto com grande *overhead*, o que desaconselha seu uso em sistemas 6LoWPAN. No entanto, o HTTP pode ainda ser muito útil para a comunicação entre redes 6LoWPAN e a Internet e por este motivo, a indústria e a comunidade desenvolveram uma camada de protocolos de aplicação alternativa denominada CoAP, que é um protocolo de mensagem que funciona sobre o UDP e com um mecanismo REST (*Representational State Transfer*) otimizado, muito semelhante ao HTTP. O CoAP é definido pela IETF na [RFC7252] e define as retransmissões, tipos de mensagens confirmáveis e não confirmáveis, suporte à dispositivos que entram em modo "*sleep*", transferências de blocos, suporte a subscrição e descoberta de recursos. O CoAP também é facilmente mapeado para HTTP por intermédio de *proxies*. Outros protocolos de aplicação, como o MQTT (*Message Queue Telemetry Transport*) ou *Websocket* têm sido citados na literatura [OLS2014] como opções de protocolos de aplicação para uso em RSSFs, porém não serão considerados nesse estudo, que tem como base o protocolo CoAP.

2.5.1. Protocolo CoAP

O modelo de interação do CoAP é do tipo Cliente-Servidor, com comunicação assíncrona por meio de mensagens UDP. Estas mensagens são de quatro tipos: "confirmáveis", "não confirmáveis", de "confirmação de recebimento" e de "*Reset*". Os métodos empregados pelo CoAP são similares aos utilizados no protocolo HTTP: GET, POST, PUT e DELETE. A Figura 2.15 mostra o posicionamento do CoAP, entre a camada de transporte UDP e a aplicação.

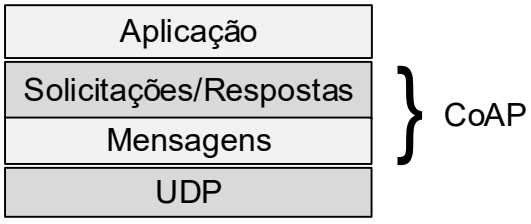

Figura 2.15 – Representação em camadas do CoAP [RFC7252]

O formato das mensagens CoAP é composto de um cabeçalho fixo de 4 bytes, com campos opcionais no formato TLV (*Type-Length-Value*) mostrado na Figura 2.16.

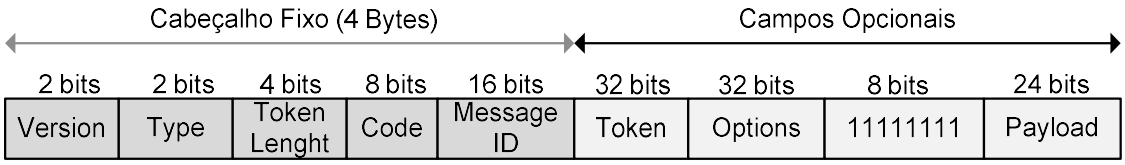

Figura 2.16 – Formato do quadro CoAP [RFC7252]

Quando é enviada uma requisição do tipo "confirmável", indicada pelo campo "*Type*" igual a "0", é obrigatório o recebimento de uma mensagem de confirmação "Ack", indicada pelo campo "Type" igual a "2". Assim é possível se simular uma conexão orientada a conexão mesmo se utilizando um protocolo de transporte não orientado à conexão, como é o caso do UDP. Isto permite uma comunicação confiável em um meio sujeito a perdas, como é o caso do meio sem fio.

Uma mensagem "*Reset*" (campo "*Type*" igual a "3") indica que uma mensagem específica ("confirmável" ou "não confirmável") foi recebida, mas falta algum contexto para processá-lo corretamente. Esta condição é geralmente causada quando o nó de destino foi reiniciado ou houve a perda de algum estado que seria necessário para interpretação correto da mensagem. O envio de uma mensagem de "*Reset*" também é uma forma simples para a verificação da existência de um nó, também denominada de "CoAP-Ping".

Outro ponto a se notar é que as mensagens enviadas via 6LoWPAN são pequenas o suficiente para caberem em um único pacote IEEE 802.15.4, evitando assim as fragmentações das mensagens em diversos pacotes.

Os principais recursos deste protocolo são citados na [RFC7252]:

- Protocolo Web que preenche os requisitos da comunicação M2M (*Machine-to-Machine*) em ambientes constritos;
- Comunicação via protocolo UDP com suporte opcional a solicitações *unicast* e *multicast* confiáveis (com confirmação de recebimento);
- Troca de mensagens assíncronas;
- Pequeno *overhead* de cabeçalho e baixa complexidade de interpretação;
- Suporte a URI (*Uniform Resource Identifiers*) e *Content-Type*;
- Capacidade de Proxy e *caching*;

- Mapeamento HTTP *stateless*, permitindo a construção de *proxies* que proporcionam acesso a recursos CoAP via HTTP de uma forma uniforme ou por simples interfaces HTTP;
- Opção de segurança por meio do DTLS [RFC6347].

As mensagens CoAP seguem o estilo de arquitetura de *software* REST, onde os mesmos verbos são utilizados para busca e inserção de dados em servidores remotos por meio de serviços *web*. Os sistemas REST interagem com sistemas externos por meio de URIs com formato definido na [RFC3986]. A estrutura de URI CoAP segue o modelo definido nesta RFC:

```
coap-URI = "coap:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

O parâmetro "*host*" refere-se ao nome do servidor CoAP, que pode ser resolvido por um serviço de resolução de nomes como o DNS (*Domain Name Service*) ou o endereço IPv6 diretamente, "*port*" refere-se à porta UDP em que o servidor está ouvindo, "*path*" define qual o recurso que está sendo acessado no servidor e a "*query*", no formato de pares "chave=valor", permite o envio de parâmetros para o recurso.

A [RFC7252] define em sua Seção 5.8 os métodos a serem utilizados nas mensagens de requisição:

- GET: busca uma representação para a informação que corresponde ao recurso identificado pela URI de requisição;
- POST: requisita que a representação inclusa na requisição seja processada. Normalmente resulta na criação de um novo recurso ou na atualização do recurso de destino;
- PUT: requisita que a o recurso identificado pela URI seja atualizado ou criado pela representação incluída na URI. Se o recurso existe, a representação incluída pode ser considerada uma versão modificada do recurso e um código de resposta 2.04 (Modificado) pode ser retornado. Se o recurso não existir, o servidor deve criar o recurso e deve retornar uma um código de resposta 2.01 (Criado).
- DELETE: requisita que o recurso identificado pela URI de requisição seja apagado, com a geração opcional de um código de retorno 2.02 (Removido).

Para o retorno das mensagens de medidas foi adotado no protocolo SD6WSN o formato de dados JSON (*JavaScript Object Notation*), definido na [RFC4627]. A IANA (*Internet Assigned Numbers Authority*) definiu para o JSON um *subtype name* como application/json.

A Figura 2.17 exemplifica uma topologia de comunicação entre os servidores CoAP instalados em nós de uma RSSF 6LoWPAN e um cliente CoAP, hospedado em um servidor central na rede IPv6. Pode-se observar a presença do roteador 6LBR, responsável pela ligação entre a rede IPv6 e a rede 6LoWPAN, com uma seta exemplificando a comunicação de uma mensagem CoAP POST entre o cliente e um servidor, de uma forma transparente.

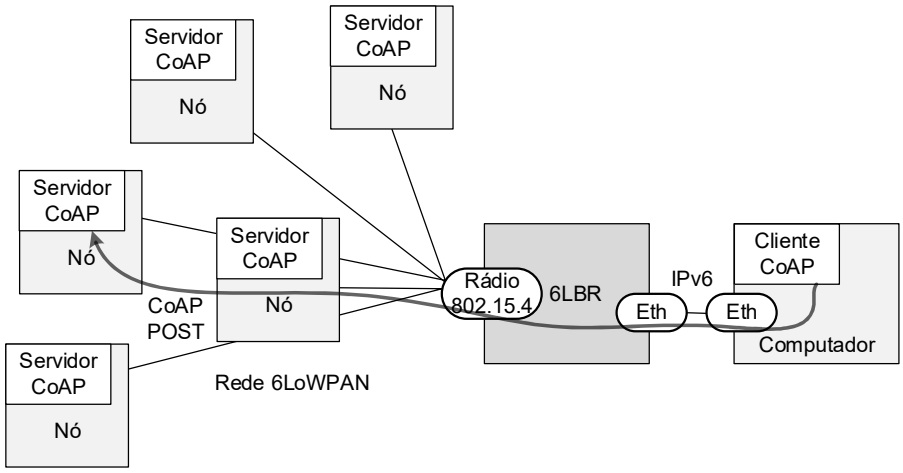

Figura 2.17 - Topologia básica de uma rede 6LowPAN com servidores CoAP

2.6. Redes Definidas por *Software*

As redes de comunicação de dados baseadas no protocolo TCP/IP se desenvolveram de uma forma orgânica, com expansões feitas pela inclusão de elementos autônomos de diferentes gerações e capacidades. Esta mescla de elementos, que vão desde *switch*es, roteadores*, firewalls* até roteadores Wi-Fi residenciais, de múltiplos fabricantes e com versões de padrões e protocolos muitas vezes conflitantes criaram redes minimamente estáveis e muitas vezes difíceis de serem gerenciadas [HAM2009]. A inclusão de novos serviços em uma rede heterogênea passou a ser uma tarefa arriscada, com uma grande probabilidade de criar instabilidades em serviços já instalados. A mesma facilidade criada pela autonomia dos elementos, de se inserirem na rede e, através de protocolos padrão tentarem se adaptar à topologia existente criou barreiras que impedem a inovação na criação de novos serviços, face às limitações de outros elementos. A falta de um padrão de configuração comum aos elementos

cria dificuldades no gerenciamento da rede como um todo, com muitos elementos podendo ser apenas configurados por especialistas e com interfaces criadas há mais de 10 anos atrás [MCK2008].

As SDNs procuram atacar as principais deficiências das redes atuais com a introdução de um novo paradigma: simplificar os elementos de rede e centralizar as decisões em um elemento controlador externo, que tomaria todas as decisões com base em sua visão única da topologia existente. Com isso, os elementos de rede realizariam apenas as tarefas para quais seu *hardware* foi projetado e simplificaria toda a camada de *software* e processamento hoje necessária para que um elemento se integre às redes existentes. Um processamento central simplificaria a criação de novos serviços em uma rede, e por sua visão completa do que acontece na rede, a probabilidade de se afetar outro serviço já existente seria drasticamente reduzida.

As SDNs seguem três princípios básicos:

- Separação da funcionalidade do plano de controle da funcionalidade de encaminhamento de dados, deixando esta segunda para dispositivos de *hardware* específicos, tais como ASICs (*Application Specific Integrated Circuits*) padronizando e simplificando suas funções;
- Um controlador centralizado com a visão detalhada da rede, que abstrai a complexidade da rede, incluindo seus dispositivos de rede, servidores e máquinas virtuais.
- Extensibilidade provida por um padrão aberto que permita a programação da rede por meio de aplicações externas.

A Figura 2.18 mostra um exemplo de uma rede tradicional do lado esquerdo e no lado direto, a mesma rede seguindo o princípio SDN.

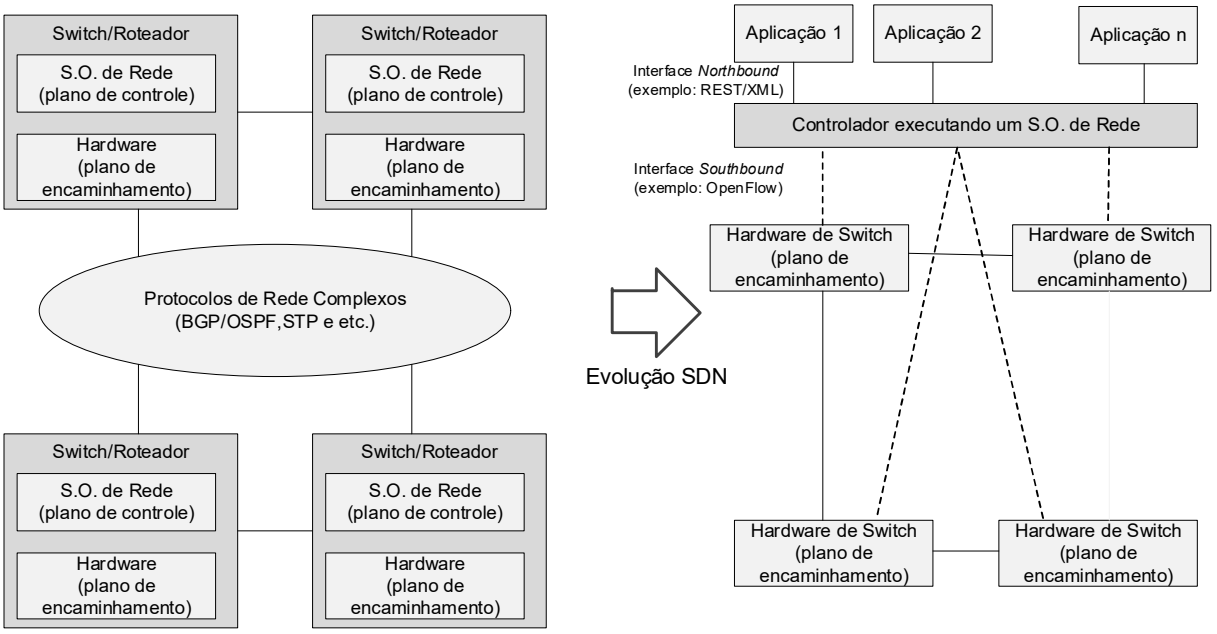

Figura 2.18 – Comparação entre redes tradicionais e SDN

Em uma SDN, a comunicação entre o controlador e os switches é feita por uma interface denominada *Southbound* via um protocolo padrão, como o *OpenFlow*, e a comunicação das aplicações com o controlador é realizada pela interface *Northbound* por diversas APIs (*Application Programming Interface*), tais como o XML ou o REST.

2.6.1. Protocolo OpenFlow

O protocolo *OpenFlow* foi fruto de um desenvolvimento conjunto de diversos esforços para a criação de redes programáveis durante a década de 2000 [MCK2008]. O *OpenFlow* teve como seu antecessor o projeto SANE/Ethane da Universidade de Stanford, que em 2006 definiu uma nova arquitetura para redes empresariais. Mais tarde um grupo de operadoras, prestadores de serviços e fabricantes criou a ONF (*Open Networking Foundation*) para promover as redes SDN e padronizar o protocolo *OpenFlow*.

A arquitetura básica de uma SDN é formada pelo switch *OpenFlow*, o controlador SDN, com as interfaces *Southbound OpenFlow* e a interface *Northbound* com as aplicações.

A comunicação entre o controlador e os switches *OpenFlow* se dava originalmente, na especificação *OpenFlow* 1.0 [ONF2010], por mensagens que trafegavam por pacotes TCP sobre TLS (*Transport Layer Security*) porém nas especificações seguintes foi retirada a obrigatoriedade do TLS [BEN2013].

2.6.1.1. Tipos de mensagens OpenFlow

A comunicação entre o controlador SDN e os *switch*es *OpenFlow* é realizada por três tipos de mensagens:

- Controlador aos nós: mensagens iniciadas pelo controlador e que podem ou não requerer uma resposta;
- Assíncronas: mensagens originadas pelo *switch* quando da alteração de algum parâmetro monitorado, ou pela chegada de um novo pacote que não faz parte da tabela de fluxos;
- Simétricas: tanto o *switch* quanto o controlador podem enviar mensagens para o outro sem que haja uma solicitação pelo outro lado.

As mensagens definidas na especificação *OpenFlow* 1.0 [ONF2010] estão transcritas na Tabela 2.3.

Tabela 2.3 – Mensagens *OpenFlow* 1.0 [ONF2010]

OpenFlow Packet Type	Classification of Packet	Type Value in OF Generic Header
Symmetric Messages		
OpenFlow hello	Symmetric message	0
OpenFlow error	Symmetric message	1
OpenFlow echo request	Symmetric Message	2
OpenFlow echo reply	Symmetric message	3
OpenFlow vendor message	Symmetric message	4
Switch Configuration Messages		
OpenFlow feature request	Controller-to-switch message	5
OpenFlow Feature reply	Controller-to-switch message	6
OpenFlow get configuration request	Controller-to-switch message	7
OpenFlow get configuration reply	Controller-to-switch message	8
OpenFlow set configuration	Controller-to-switch Message	9
Asynchronous Messages		
OpenFlow packet-in	Asynchronous message	10
OpenFlow flow removed	Asynchronous message	11
OpenFlow port status	Asynchronous message	12
Controller command Messages		
OpenFlow packet-out	Controller-to-switch message	13
OpenFlow flow modification	Controller-to-switch message	14
OpenFlow port modification	Controller-to-switch message	15
Statistics Messages		
OpenFlow stats request	Controller-to-switch message	16
OpenFlow stats reply	Controller-to-switch message	17
Barrier Messages		
OpenFlow barrier request	Controller-to-switch message	18
OpenFlow barrier reply	Controller-to-switch message	19
Queue Configuration Messages		
Queue get configuration request	Controller-to-switch message	20
Queue get configuration reply	Controller-to-switch message	21

2.6.2. Tabela de fluxos

Esta seção descreve os componentes da tabela de fluxos e o processo em que os pacotes que adentram no *switch* são comparados com as entradas da tabela.

Embora existam *switches* híbridos que operam da forma tradicional em conjunto com o protocolo *OpenFlow*, um switch *OpenFlow* é formado por uma tabela de fluxos, a qual realiza as funções de inspeção e encaminhamento de pacotes. Cada tabela de fluxo no switch é composta de um conjunto de entradas de fluxo, que, conforme a Tabela 2.4, consistem dos seguintes campos [ONF2010]:

- Campos de cabeçalho ou campos de *match*, com informação encontrada no cabeçalho do pacote, a porta de ingresso do pacote e metadados, usados para identificar os pacotes coincidentes de entrada;
- Contadores, utilizados para coletar estatísticas de um fluxo em particular, tais como número de pacotes recebidos, duração do fluxo ou número de bytes transferidos;
- Um conjunto de ações a serem aplicadas após uma identificação de um pacote coincidente.

Tabela 2.4 – Composição de uma entrada na tabela de fluxos [ONF2010]

Campos de cabeçalho	Contadores	Ações

2.6.2.1. Campos de cabeçalho

O campo “Campos de cabeçalho” possui a informação de como os pacotes de entrada serão comparados e pode conter um valor especifico de “*match*” ou a palavra-chave “ANY” que significa qualquer valor.

A avaliação de um novo pacote entrando em um *switch* é realizada comparando-se o cabeçalho do pacote com uma estrutura de *match*, onde certos campos são avaliados de acordo com as entradas da tabela de fluxos. A estrutura com os campos de *match* da versão *OpenFlow* 1.0 [ONF2010] pode ser vista na Figura 2.19, observando-se que não há campos para o protocolo IPv6.

N bits	48 bits	48 bits	16 bits	12 bits	3 bits	32 bits	32 bits	8 bits	6 bits	16 bits	16 bits
Ingress Port	Src MAC	Dst MAC	Eth Type	VLAN ID	VLAN PRI	Src IP	Dst IP	IP Proto	TOS Bits	Transport Src Port	Transport Dst Port

Figura 2.19 - Estrutura de campos de *match* da versão *OpenFlow* 1.0 [ONF2010]

A descrição de cada um dos campos de *match* é apresentada na Tabela 2.5.

Tabela 2.5 - Campos de *match* da versão *OpenFlow* 1.0 [ONF2010]

Field	Bits	When applicable	Notes
Ingress Port	(Implementation dependent)	All packets	Numerical representation of incoming port, starting at 1.
Ethernet source address	48	All packets on enabled ports	
Ethernet destination address	48	All packets on enabled ports	
Ethernet type	16	All packets on enabled ports	An OpenFlow switch is required to match the type in both standard Ethernet and 802.2 with a SNAP header and OUI of 0x000000. The special value of 0x05FF is used to match all 802.3 packets without SNAP headers.
VLAN id	12	All packets of Ethernet type 0x8100	
VLAN priority	3	All packets of Ethernet type 0x8100	VLAN PCP field
IP source address	32	All IP and ARP packets	Can be subnet masked
IP destination address	32	All IP and ARP packets	Can be subnet masked
IP protocol	8	All IP and IP over Ethernet, ARP packets	Only the lower 8 bits of the ARP opcode are used
IP ToS bits	6	All IP packets	Specify as 8-bit value and place ToS in upper 6 bits.
Transport source port / ICMP Type	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Type
Transport destination port / ICMP Code	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Code

Para poder acomodar um número maior de campos de *match*, inclusive os relativos ao protocolo IPv6, a partir da versão 1.2 do protocolo *OpenFlow* houve uma mudança na estrutura de *match* com a introdução da OXM (*OpenFlow Extensible Match*). Uma regra de *match* passou a ser definida por uma sequência de OXMs, onde todas devem se obedecidas. Os OXMs são segregados por classe e em seguida por campo. Atualmente, existe somente uma classe a ser considerada, denominada "*OpenFlow Basic*", identificada pelo valor de classe 0x8000. O campo de comprimento de um OXM é único e somente especifica o número de bytes no *payload* OXM. O cabeçalho OXM tem um único bit que indica se a máscara de bits está presente no *payload*. Se o *payload* estiver vazio, significa que tudo é aceito. A Figura 2.20 mostra a estrutura de *match* para as versões de *OpenFlow* a partir da 1.2.

Pela complexidade da estrutura do *OpenFlow* 1.2, no presente trabalho adotou-se a estrutura da versão 1.0, com a mudança dos campos de *match* relativos ao protocolo IPv4 para o IPv6, com a consequente alteração dos tamanhos de 32 para 128 bits e Ethertype 0x0800 para 0x86dd.

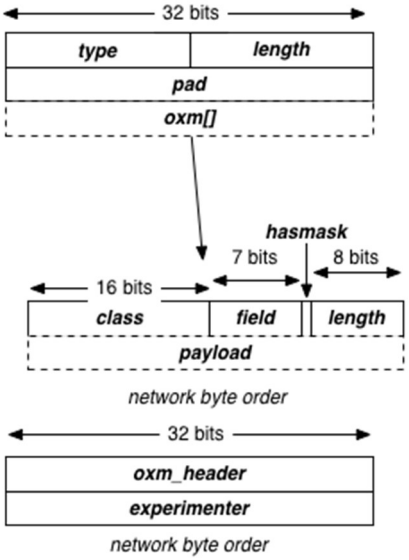

Figura 2.20 - Estrutura de *match* da versão *OpenFlow* 1.2 [ONF2013]

2.6.2.2. Ações

Cada entrada de fluxo é associada com nenhuma ou mais ações, que ditarão como o *switch* deve tratar os pacotes que obedecerem ao *match*. Se não houver uma ação de encaminhamento, o pacote é descartado. As ações devem obedecer à mesma sequência das entradas dos fluxos.

Um *switch* não precisa suportar todos os tipos de ações, somente as requeridas. As ações requeridas para a ação *forward* (encaminhamento) [ONF2010] são:

- ALL: envia o pacote para todas as interfaces, inclusive para a que o pacote ingressou;
- CONTROLLER: encapsula e envia o pacote para o controlador;
- LOCAL: envia o pacote para a pilha de rede local do *switch*;
- TABLE: realiza ações na tabela de fluxos. Apenas para mensagens *packet-out*;
- IN-PORT: envia o pacote para a porta de entrada.

A ação requerida *drop* (descarte) é utilizada quando não há ação definida para todos os pacotes que se enquadrarem em determinada regra de *match*.

Apesar de não ser estritamente requerida, as ações *Modify* constantes na Tabela 2.6 são de grande utilidade em uma implementação *OpenFlow*.

Tabela 2.6 - Ações *Modify* do *OpenFlow* 1.0 [ONF2010]

Action	Associated Data	Description
Set VLAN ID	12 bits	If no VLAN is present, a new header is added with the specified VLAN ID and priority of zero. If a VLAN header already exists, the VLAN ID is replaced with the specified value.
Set VLAN priority	3 bits	If no VLAN is present, a new header is added with the specified priority and a VLAN ID of zero. If a VLAN header already exists, the priority field is replaced with the specified value.
Strip VLAN header	-	Strip VLAN header if present.
Modify Ethernet source MAC address	48 bits: Value with which to replace existing source MAC address	Replace the existing Ethernet source MAC address with the new value
Modify Ethernet destination MAC address	48 bits: Value with which to replace existing destination MAC address	Replace the existing Ethernet destination MAC address with the new value.
Modify IPv4 source address	32 bits: Value with which to replace existing IPv4 source address	Replace the existing IP source address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applicable to IPv4 packets.
Modify IPv4 destination address	32 bits: Value with which to replace existing IPv4 destination address	Replace the existing IP destination address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applied to IPv4 packets.
Modify IPv4 ToS bits	6 bits: Value with which to replace existing IPv4 ToS field	Replace the existing IP ToS field. This action is only applied to IPv4 packets.
Modify transport source port	16 bits: Value with which to replace existing TCP or UDP source port	Replace the existing TCP/UDP source port with new value and update the TCP/UDP checksum. This action is only applicable to TCP and UDP packets.
Modify transport destination port	16 bits: Value with which to replace existing TCP or UDP destination port	Replace the existing TCP/UDP destination port with new value and update the TCP/UDP checksum This action is only applied to TCP and UDP packets.

2.6.2.3. Contadores

Os contadores são mantidos por tabela, por fluxo, por porta e por fila. A Tabela 2.7 apresenta a lista dos contadores requeridos.

Tabela 2.7 - Contadores do *OpenFlow* 1.0 [ONF2010]

Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

2.6.3. Arquitetura básica de uma rede SDN OpenFlow

Os componentes básicos de uma rede SDN *OpenFlow* são o controlador e os *switches OpenFlow* e uma rede que representa a arquitetura básica SDN é apresentada na Figura 2.21.

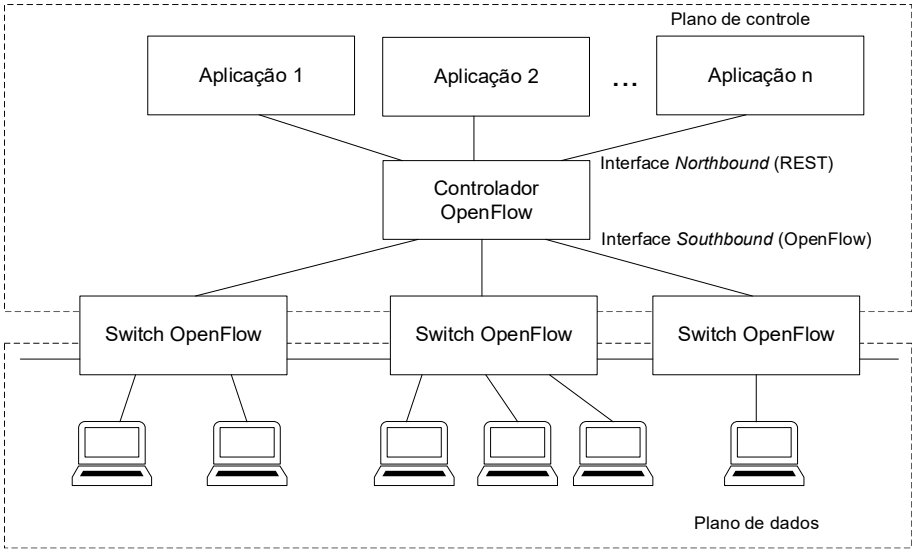

Figura 2.21 – Exemplo de arquitetura *OpenFlow*

A comunicação entre a interface *Southbound* do controlador *OpenFlow* e os *switches* é realizada pela rede TCP/IP do denominado "plano de controle". Por ela trafegam as mensagens do protocolo *OpenFlow*.

Pela interface *Northbound*, as aplicações tais como comutação de pacotes (*switching*), *firewalls*, balanceadores de tráfego e etc. se comunicam com o controlador por mensagens REST ou XML, dando instruções a este de como tratar os fluxos de dados.

Com as mensagens recebidas do controlador, os *switches* criam entradas nas suas tabelas de fluxo, que disciplinam o tráfego do plano de encaminhamento, representado na Figura 2.21 pelos computadores ligados aos *switches*.

Quando um pacote ingressa em um *switch*, o cabeçalho deste pacote tem alguns de seus campos comparados com a tabela de fluxos. Se já existir uma regra estabelecida para aquele pacote pertencente a um fluxo, uma ação como descarte, encaminhamento para uma porta ou mudança no cabeçalho é efetuada. Caso não exista esta entrada na tabela, o conteúdo do cabeçalho é enviado para o controlador por meio de uma mensagem *Packet-in* do protocolo *OpenFlow*, para que uma regra instalada por uma aplicação defina uma ação, posteriormente comunicada para o *switch* por uma mensagem *OpenFlow* de resposta.

2.7. Conclusão

Neste capítulo foram abordadas as principais tecnologias empregadas no desenvolvimento do presente trabalho, iniciando-se com os padrões das camadas física e MAC definidos pelo IEEE, a camada de adaptação 6LoWPAN, o protocolo de roteamento RPL que é o adotado pelo IETF para RSSFs do tipo 6LoWPAN e o protocolo de aplicação CoAP, utilizado como transporte das mensagens do protocolo SD6WSNP, apresentado no Capítulo 4. Também foram apresentados o conceito SDN e o protocolo OpenFlow, que forneceram as bases para a definição do *framework* SD6WSN.

No próximo Capítulo serão discutidos os trabalhos relacionados à presente tese e as diferentes abordagens propostas para as RSSFs definidas por *software*.

Capítulo 3

Trabalhos Relacionados

Esse capítulo descreve os trabalhos que avaliam o protocolo RPL, especialmente aqueles onde o ambiente utilizado é o Contiki/COOJA e pesquisas que propõem o uso da abordagem SDN em redes de sensores sem fio.

3.1. Avaliação do RPL

No artigo de autoria de Clausen *et al.* [CLA2011] são discutidas diversas questões críticas sobre a concepção e o funcionamento do RPL e que podem causar um funcionamento não otimizado da RSSF, tais como:

- O DODAG do RPL é formado tendo como premissa que o tráfego é *convergecast* na maior parte dos cenários, com uma probabilidade de tráfego muito baixa entre os sensores. Segundo o artigo, o tráfego entre sensores não é tão raro e que quando ocorrer, se o RPL operar no *non-storing-mode*, todo o tráfego deverá passar pela raiz ou se for *storing-mode* o tráfego passa pelo ancestral comum. Isto pode causar congestionamentos nos enlaces próximos à raiz, principalmente em redes operando no *non-storing-mode*;
- As mensagens DIO podem superar os 79 bytes úteis de *payload* de camada 3 disponíveis nas mensagens ICMPv6, o que provoca a fragmentação das mensagens DIO, que devem ser enviadas em dois pacotes;
- Questões sobre o mecanismo DAO, onde o RPL implementa dois modos de operação incompatíveis entre si. Além de possível fragmentação, quando se utiliza *source-routing* há um limite de oito saltos (sem compressão de endereços) ou 64 saltos (com

compressão), o que limita sua utilização em topologias lineares. Quando o *storing-mode* é empregado, há a necessidade que os nós próximos à raiz tenham uma capacidade de memória elevada, para armazenar todas as rotas presentes em seu sub-DODAG;

- Considerações relativas à agregação de rotas com o intuito de diminuição das tabelas de roteamento dos nós, principalmente daqueles próximos à raiz. São discutidos os impactos negativos da adoção desta técnica, comum em redes IP em geral, mas que em redes RPL trariam problemas como limitação de número de pais alternativos, perda de conectividade quando do recálculo da rede e possível conexão de nós em sub-redes diferentes à que a rota agregada aponta;
- Considerações quanto à hipótese da bidirecionalidade dos enlaces, pois a rota de subida, ou seja, dos nós até a raiz, é determinada por mensagens DIO, que fluem da raiz para os nós. O RPL assume que os enlaces são perfeitamente bidirecionais, o que não é verdade para muitos casos;
- Ainda sobre a questão da unidirecionalidade dos enlaces, os autores argumentam porque o NUD (*Neighbor Unreachability Detection*) não é uma solução, pois começam a agir após o DODAG estar montado com a premissa de que os enlaces são bidirecionais.
- Complexidade de implementação do protocolo RPL em dispositivos embarcados típicos em redes LLN, que possuem poucos recursos de memória e processamento;
- Subespecificação de alguns mecanismos do RPL como a temporização das mensagens DAO e DAO-ACK, que podem ocasionar baixo desempenho da rede se não forem cuidadosamente dimensionados. O mecanismo de reparo local também pode causar *loops* e perda de pacotes se não for corretamente implementado;
- O algoritmo *Trickle* não apresenta desempenho otimizado em cenários reais, onde a qualidade dos enlaces é variável, com a consequente emissão de mensagens DIO com frequência maior do que a esperada e causando aumento de tráfego de controle;
- O RPL não garante a ausência de *loops* nem um tempo de convergência determinado, porém pode reparar um *loop* quando ele é detectado. Os *loops* são frequentes em implementações do RPL como no ContikiRPL e os reparos podem aumentar os atrasos na rede, bem como o aumento de tráfego de controle. No "*non-storing-mode*" o problema é mais acentuado, pois é necessário um reparo global para a reconstrução do DODAG.

O artigo de Ancillotti *et al.* [ANC2012] discute o desempenho do protocolo RPL em redes AMI. A ênfase é dada nas características de confiabilidade e estabilidade das redes RPL e no fato constatado pelos autores que o RPL frequentemente escolhe rotas subotimizadas por caminhos de baixa qualidade devido a decisões de roteamento ineficientes. Estas decisões de roteamento são o principal fator de diminuição das taxas de sucesso de encaminhamento de pacotes. Também são apontados fatores como o pouco conhecimento pelo RPL dos parâmetros de qualidade dos enlaces. As simulações foram efetuadas utilizando-se a versão da implementação RPL do Contiki disponível na época e os autores propõem a investigação de novos mecanismos que corrijam os problemas encontrados como trabalhos futuros.

Em um trabalho seguinte [ANC2013], os mesmos autores analisam o papel do protocolo RPL em um contexto mais amplo na comunicação de redes *Smart Grid*, concebidas para automação de redes elétricas. No texto são apontadas as mesmas restrições encontradas no artigo anterior, mas em comparação às demais tecnologias existentes, tais como a LTE (*Long Term Evolution*), em diversos quesitos, tais como custo e escalabilidade, os autores concluem que a aplicação do RPL é viável para a função de AMI dentro de redes *Smart Grid*.

Em sua dissertação de Mestrado, Ali [ALI2012] faz um extensivo estudo de comparação das "Funções Objetivo" existentes para o RPL, a "OF0", baseada no número de saltos entre os nós, e o "ETX", que utiliza as métricas de qualidade dos enlaces para o cálculo da posição dos nós na rede. Para ambas as "Funções Objetivo", é avaliado o impacto dos parâmetros do RPL no desempenho de uma RSSF. As métricas utilizadas para as comparações foram latência, PDR (*Packet Delivery Ratio*), controle de tráfego, tempo de convergência e consumo de energia. Na conclusão do trabalho, o autor concluiu que a "Função Objetivo" ETX apresenta melhores resultados em ambientes sujeitos a perdas e foram apresentados os valores ideais para os parâmetros RPL "DIO *Interval Min*", "DIO *Interval Doubling*", "RDC (*Radio Duty Cycling*) *Rate*" e a frequência de mensagens de aplicação.

Gaddour *et al.* [GAD2012] analisam o impacto dos parâmetros do RPL e as "Funções Objetivo" OF0 e ETX no desempenho de formação da rede em termos de consumo de energia, de acréscimo na utilização de memória dos nós, no aumento de tráfego de controle, no tempo de convergência da rede e no máximo número de saltos. A análise foi feita em ambiente simulado COOJA com nós Contiki. Os autores concluíram que a "Função Objetivo" OF0 oferece menor consumo de energia e menor número de saltos, mas não leva em conta o desempenho dos enlaces.

Zhang e Li [ZHA2014] apresentam as análises realizadas no simulador COOJA de uma rede RPL de vinte nós em função dos parâmetros de consumo de energia, de latência e tempo de convergência. O tempo de convergência foi analisado em função da densidade da rede, do número de saltos e da quantidade de DODAGs. Também foi analisada a variação do número de mensagens de controle RPL, com o aumento do número de nós para cenários simulados de até noventa nós. A conclusão sobre o consumo de energia e sobre o tempo de convergência é que estes fatores podem ser reduzidos com a diminuição de mensagens DAO provido por redes com menor número de saltos e com diversos nós raiz.

Dawans *et al.* [DAW2012] discutem em seu artigo as dificuldades existentes nas redes RPL em se estimar a qualidade dos enlaces e do gerenciamento das políticas de vizinhança entre nós em ambientes de alta densidade. Para a avaliação, foram desenvolvidas diversas técnicas de teste de qualidade de enlaces e políticas de gerenciamento de vizinhança na implementação RPL do S.O. Contiki. A abordagem proposta para a estimativa de qualidade de enlaces foi a "otimista", onde se assume que se um novo vizinho for detectado, será atribuída a ele uma métrica ETX=1 (a melhor possível). Se ocorrerem retransmissões excessivas, este valor é gradativamente aumentado até o ponto que outro vizinho seja escolhido. Para a política de gerenciamento de vizinhança, os autores propuseram a criação de uma tabela com os vizinhos que apresentaram o pior desempenho, evitando assim futuras reavaliações de qualidade de enlace.

Kathuria *et al.* [KAT2013] apresentam em seu artigo um estudo comparativo entre os protocolos RPL e AODV (*Ad hoc On-Demand Distance Vector Protocol*) aplicados em redes de medição inteligente de energia. As simulações efetuadas concluíram que o protocolo AODV, considerado padrão para redes *ad hoc* e *mesh* apresenta desempenho inferior em termos de latência e PDR às redes baseadas em RPL com um grande número de nós, como as de medição de energia.

Herberg e Clausen [HER2011] comparam em seu estudo o RPL com o protocolo LOAD (*6LoWPAN Ad Hoc On-Demand Distance Vector Routing*), derivado do AODV em redes LLN de grande porte e com a predominância de tráfego bidirecional. Os autores argumentam que o RPL é otimizado para redes multiponto-a-ponto, com raro tráfego ponto-a-multiponto e que a comunicação entre os nós é praticamente nula. Porém, em redes AMI, há a necessidade dos medidores de energia ou controladores de carga receberem comandos frequentemente e que estes tipos de tráfego não podem ser desprezados. As simulações foram realizadas no simulador

NS-2 com até 1000 roteadores IEEE 802.11b em uma densidade de 50 roteadores por km^2. Os resultados para este cenário mostraram que o RPL proporciona menor latência no encaminhamento de pacotes e o LOAD introduz na rede menor tráfego de controle.

Yi *et al.* [YI2013] fazem uma comparação do RPL com o protocolo LOADng (*Lightweight On-demand Ad hoc Distance-vector Routing*), sucessor do LOAD, que teve seu desenvolvimento suspenso pelo grupo de trabalho 6LoWPAN. O cenário escolhido para a simulação foi de redes AMI com um número de roteadores IEEE802.11 que variou de 63 a 500. A conclusão do trabalho aponta diversas limitações do RPL, que foi desenhado para redes mutiponto-a-ponto e que não é apropriado para redes que tenham comunicação entre os nós (ponto-a-ponto). Há uma dependência muito grande em relação ao nó raiz da rede, com um potencial de ser um ponto crítico em caso de falhas e de gargalo de tráfego. Segundo os autores, as redes roteadas pelo RPL tem uma alta tendência de *loops* e de fragmentação de pacotes. Também é apontada a falta de um mecanismo de teste para enlaces unidirecionais, já que as rotas de descida (da raiz aos nós) são estabelecidas em função das rotas de subida. Já o LOADng não permite *loops* e possui suporte a enlaces unidirecionais, mas por ser um protocolo reativo, pode haver um atraso na fase de descoberta de rotas e apresenta maior tráfego de controle se houver muitos fluxos de tráfego entre sensores na rede.

Long *et al.* [LON2012] descrevem em seu trabalho uma comparação de desempenho realizada entre o RPL e o CTP (*Collection Tree Protocol*) em ambiente simulado COOJA. As métricas para a avaliação foram o PRR, consumo de energia por pacote, número de pacotes que estiveram em *loop*, número de trocas de nó "pai" e tempo de convergência da rede. Os resultados mostraram pequena vantagem para o CTP em redes de 9 nós, porém em redes de 25 e 49 nós, o RPL mostrou melhor desempenho em termos de maior PRR e consumo de energia, principalmente em situações de maior tráfego.

Mazzer *et. al.* [MAZ2013] comparam as diferentes implementações dos subconjuntos 6LoWPAN para várias plataformas de nós sensores, incluindo *hardware* e sistemas operacionais. Os autores identificaram cinco sistemas operacionais que implementam a pilha 6LoWPAN: TinyOS, Contiki, FreeRTOS, RioT e Mantis. Pediredla *et. al.* [PED2013] descrevem uma implementação 6LoWPAN para dispositivos de capacidade restrita baseados em *software* aberto, incluindo o Contiki. Os autores incluíram características importantes como compressão de cabeçalho e endereçabilidade de IP, mas na época, apenas a topologia em estrela era suportada, e uma versão simplificada da descoberta do vizinho foi implementada.

3.2. Abordagem SDN em RSSFs

O controle de encaminhamento de pacotes em RSSFs por meio de técnicas SDN foi abordado em alguns trabalhos relacionados a seguir.

O artigo de Muhamani [MAH2011] foi o primeiro a explorar o tema SDN em RSSFs. A proposta foi de empregar o protocolo *OpenFlow* em redes de sensores, e segundo os autores, simulações demonstraram que os nós *flow-sensors* são muito mais confiáveis do que os nós convencionais, pois os pacotes de dados e de controle, além dos nós em si podem ser monitorados e roteados quando necessário. O desempenho apresentado nas simulações foi similar ou superior ao das redes de sensores convencionais.

Luo *et al.* [LUO2012] apresentam em seu artigo proposições para endereçar os desafios de se utilizar o *OpenFlow* em RSSFs, já que este foi desenhado para redes de alta velocidade cabeadas. Os desafios citados são:

- A criação de fluxos no plano de dados, a criação do canal de controle entre os nós e o controlador SDN;
- A questão do tráfego de controle compartilhado com o de dados, já que utilizam o mesmo meio de transmissão;
- A geração de tráfego pelos próprios nós, que atuam como sensores também;
- A necessidade de processamento de dados nos nós, com o intuito de diminuir a utilização de banda da rede;
- Compatibilidade com nós existentes não *OpenFlow*.

Os autores propõem utilizar o *OpenFlow* 1.3 em RSSFs Zigbee, e, portanto, em camada dois ou com o 6LoWPAN, porém em topologia *mesh-under*, que não utiliza o protocolo RPL. O artigo, como sendo um dos primeiros a abordar o paradigma SDN para as RSSFs, concentrou-se na apresentação dos desafios, como norteador para desenvolvimentos futuros.

Costanzo *et al.* [COS2012] apresentam uma proposta mais completa do que a apresentada por [LUO2012], incluindo alguns recursos importantes para RSSFs, não citados naquele. Na arquitetura proposta são definidos dois tipos de nós, os "genéricos", onde as tabelas de fluxo são instaladas e o *sink*, onde estão presentes duas aplicações, o controlador SDN e um virtualizador de rede. Também enumeram algumas ações pertinentes para as SDWN (*Software Defined Wireless Networks*), tais como: "*forward*", "*modify*", "*drop*", "*aggregate*" e "*turn off radio*". Por se tratar de um artigo que se propôs a apresentar a arquitetura denominada pelos

autores de “SDWN” e de relacionar as oportunidades e os desafios da aplicação do paradigma SDN em redes IEEE 802.15.4, não foram apresentados dados de desempenho, que seriam tema de trabalhos futuros.

Oliveira *et al*. [OLI2015] apresentam um *framework* SDN denominado TinySDN para RSSFs baseadas no TinyOS, com o controle de nós de sensores SDN realizado por múltiplos controladores SDN. Os nós denominados “*TinySDN-enabled*” classificam os pacotes em diversos fluxos e desempenham as duas ações especificadas, de encaminhamento ou de descarte de acordo com uma tabela de fluxo de dados. Uma segunda tabela controla o encaminhamento de pacotes de acordo com o nó de destino. A coleta de informações sobre a topologia é feita via o protocolo CTP, que por sua vez utiliza o ETX para estimar a qualidade de sinal entre os vizinhos de um nó.

Em um artigo posterior, Oliveira *et al*. [OLI2015b] discutem a substituição em trabalhos futuros do protocolo CTP pelo RPL em sua arquitetura *TinySDN*, pela capacidade do RPL em trabalhar com padrões de tráfego importantes para o campo do IoT, tais como ponto-a-ponto e ponto-a-multiponto. O artigo aponta os pontos fortes e fracos do RPL em relação ao CTP e deixa como uma possibilidade futura o estudo mais aprofundado desta alternativa.

O artigo de Gallucio *et al*. [GAL2015] apresenta em seu projeto denominado SDN-WISE alguns mecanismos simples para a definição e manuseio da tabela de fluxos que a torna “*stateful*”, com o objetivo de reduzir a quantidade de informações trocadas entre os nós sensores e o controlador de rede SDN, e tornar os nós sensores programáveis como máquinas de estados finitos, permitindo-lhes executar operações que não podem ser suportadas por soluções sem estado.

Kim *et al.* [KIM2015] propõem uma arquitetura SDN para redes de medidores de energia AMI denominada CoAP-SDAN (*Constrained Application Protocol based Sofware-Defined AMI Networking*). As simulações feitas considerando-se redes Wi-Fi, comparam a proposta com soluções de roteamento baseadas em AODV e DSDV (*Destination-Sequenced Distance Vector Protocol*).

Poucos estudos especificam o plano de controle para SDWSN, como o TinySDN [OLI2015] e CoAP-SDAN [KIM2015]. Ao contrário desses dois, o presente trabalho apresenta uma especificação detalhada de um plano controle baseado em CoAP, incluindo a especificação do protocolo de plano de controle, a estrutura da tabela de fluxo e os mecanismos de

encaminhamento no plano de dados, descoberta de topologia e controle de fluxos no controlador.

A Tabela 3.1 apresenta um resumo dos trabalhos descritos nessa Seção, com os principais aspectos de cada um.

Tabela 3.1 – Trabalhos relacionados com a abordagem SDN em RSSFs

Referência	Arquitetura	Implementação	Foco
MAH2011	Flow-Sensor	Simulação	Confiabilidade
LUO2012	Sensor-Openflow	Simulação e *testbed*	Definição do protocolo *Southbound* para redes SDN L2 (Zigbee).
COS2012	SDWN	Proposto	Dois tipos de nós, os genéricos e o *sink*. Definiu ações para os pacotes.
OLI2015	TinySDN	Simulação e *testbed*	Ferramentas para implantação de nós SDN no TinyOS, informações de topologia e qualidade via CTP. Múltiplos controladores.
OLI2015b	TinySDN	Proposto	Estudo da substituição do CTP pelo RPL
GAL2015	SDN-WISE	Simulação e *testbed*	Redução de mensagens de controle, *stateful*, nós programáveis e com máquinas de estado finitas.
KIM2015	CoAP_SDAN	Simulação e *testbed*	Redes AMI, uso do CoAP para troca de informações e comparação com AODV e DSDV

3.3. Conclusão

Este capítulo apresentou em sua Seção 3.1 alguns trabalhos que tiveram como foco a avaliação do protocolo RPL, mostrando suas virtudes e onde poderia haver melhorias, tanto em sua especificação como na implementação em bibliotecas como a presente no sistema operacional Contiki. O RPL foi comparado, principalmente em simulações, com outros protocolos propostos para RSSFs, como o LOAD, LOADng e CTP, onde algumas oportunidades de melhorias para o RPL foram apontadas. Muitas das limitações do RPL serviram como motivação para o desenvolvimento do presente trabalho, tais como o cálculo de

rotas não otimizadas para nós da rede que não fossem a raiz do DODAG e a assunção de que os enlaces são sempre bidirecionais e, portanto, com as mesmas métricas de qualidade.

O artigo de Clausen *et. al* [CLA2011] foi escrito durante a padronização do RPL pelo grupo ROLL do IETF e nele já foram apontados pontos críticos no projeto do RPL, primeiramente por assumir que o tráfego seria primordialmente no sentido dos nós para um ponto central. Em um cenário IoT atual, a comunicação bidirecional entre todos os dispositivos é muito importante, pois os nós devem receber mensagens com mesma confiabilidade que as enviam. Nas aplicações em *Smart Grid*, por exemplo, mensagens para o controle de dispositivos são tão (ou mais) importantes do que as de medida de consumo de energia. No RPL, porém, as rotas de "descida" seguem os mesmos caminhos das escolhidas para "subida", sem haver um mecanismo que verifique se este caminho é o mais adequado.

Na arquitetura SD6WSN apresentada aqui, o tráfego das aplicações é tratado com igual importância nos dois sentidos e entre os nós adjacentes, que neste caso é feito sem que haja a necessidade de encaminhamento através de um nó "pai" comum, o que acontece no "*storing-mode*" do RPL ou mesmo para a raiz do DODAG, como no "*non-storing-mode*".

Clausen *et. al.* [CLA2011] já considerava em seu artigo como fator crítico a complexidade da implementação do RPL, que exige poder de processamento e memória elevados para tratar continuamente as mudanças de topologia em redes com um grande número de nós.

Na arquitetura SD6WSN, o plano de controle continua sendo gerido pelo RPL, porém o tráfego das aplicações é disciplinado pelo algoritmo centralizado, que tem autonomia para mudar os caminhos de cada fluxo dentro da RSSF de acordo com as regras definidas também centralizadamente e com a visão da rede como um todo. As reconstruções do DODAG, por ocorrências de *loops*, as escolhas de rotas de "descida" não otimizadas e atuação do algoritmo *Tricke* com frequência, que são outras deficiências apontadas em [CLA2011], também ficariam restritas ao plano de controle do SDWSN, não afetando diretamente o tráfego de aplicações.

Dawans *et. al.* [DAW2012] citam o caso da entrada de novos nós na rede, que poderão se tornar o caminho preferencial dos pacotes de um seguimento da rede até que o número de retransmissões altere o ETX para um valor mais condizente com sua qualidade. Esta abordagem interfere no caminho dos pacotes da aplicação até que seja determinada a métrica adequada ao enlace, fazendo-os trafegar por caminhos não otimizados. Na arquitetura SD6WSN os caminhos dos pacotes de aplicação independem das decisões do RPL, e mudanças nos mesmos

são determinadas pelo algoritmo centralizado, que pode decidir por alterar o caminho de alguns fluxos passando pelos novos nós somente após a avaliação das métricas coletadas de todos os nós e das restrições aplicadas ao algoritmo.

Os estudos com os protocolos LOAD [HER2011] e LOADng [YI2013] procuraram mostrar suas vantagens em relação ao RPL, por serem concebidos para tráfego bidirecional e por serem imunes a "*loops*", características citadas como pontos críticos do RPL em outros estudos. Nas avaliações realizadas em simulações, o desempenho destes protocolos muitas vezes não superou o do RPL e apesar das desvantagens apontadas nos artigos, o RPL foi padronizado pelo IETF e tem sido largamente utilizado em RSSFs comerciais.

A Seção 3.2 apresentou alguns trabalhos que propõem a abordagem SDN em RSSFs, porém nenhum deles especificamente para redes 6LoWPAN, com a manutenção do roteamento RPL para o plano de controle e com uma solução para a comunicação dos nós com o controlador SDN utilizando o CoAP, que é um protocolo consolidado em RSSFs, como base.

Os demais artigos encontrados sobre o tema SDN para redes de sensores delimitavam seu escopo para aplicações específicas, como AMI ou em cima de modelos teóricos. No presente trabalho procurou-se buscar uma solução mais genérica, que pudesse ser aplicada da mesma forma que as redes RPL são utilizadas, mas aproveitando as já citadas vantagens do paradigma SDN.

Capítulo 4

Framework SD6WSN

Nesse capítulo é apresentado o *framework* SD6WSN, que é composto por uma arquitetura e um protocolo de comunicação baseados no paradigma SDN. Esse *framework* tem como objetivo a determinação dos caminhos entre os nós de uma RSSF de acordo com algumas características dos pacotes que trafegam pela mesma.

Nesta arquitetura, o caminho de cada fluxo, ou seja, cada conjunto de pacotes que possuam determinadas características, é ditado por um controlador central que possui a visão completa da RSSF, incluindo seus componentes e a inter-relação entre eles. Com essa visão, o controlador pode, através de aplicações que executam algoritmos para aplicações específicas de rede, instalar regras de encaminhamento para moldar o tráfego de acordo com as características esperadas para cada tipo de tráfego. Caso uma regra para um determinado tipo de tráfego já esteja instalada na tabela de fluxos, uma nova comunicação com o controlador não é necessária.

O paradigma SDN na arquitetura definida neste trabalho está representado principalmente pela capacidade de decisão centralizada e não distribuída pelos nós da rede, como no roteamento tradicional. Desta forma, os algoritmos podem ser executados em máquinas com poder de processamento muitas vezes superior aos existentes em dispositivos com limitações de CPU, memória RAM e consumo de bateria como os encontrados em nós LLN.

Outra característica das redes SDN contemplada neste projeto é a separação dos planos de controle e de encaminhamento. No caso de RSSFs, os dispositivos típicos possuem somente um transceptor, que realizam transmissões ou recepções em uma determinada frequência por

vez. Desta forma, os planos de controle e encaminhamento devem compartilhar o mesmo canal de comunicação e a banda disponível.

Para a criação do plano de controle optou-se por se manter o roteamento RPL original das RSSF-6LoWPAN, para que as mensagens de controle do protocolo proposto nesse trabalho (SD6WSNP) possam trafegar juntamente dos pacotes do protocolo RPL. Essa decisão arquitetural permite que as funções de controle de topologia e comunicação entre os nós no nível de transporte sejam mantidas através do protocolo já empregado na rede. Essa escolha é semelhante àquela feita pela arquitetura *OpenFlow*, que usa o protocolo TCP como suporte ao protocolo *OpenFlow*.

A função de controle do plano de encaminhamento dos pacotes de dados é desempenhada por um agente SD6WSN instalado no nós, que interage com o processo de encaminhamento de pacotes do sistema operacional embarcado, que teve sua lógica alterada para que suas ações fossem determinadas pela tabela de fluxos interna a cada nó, como ocorre em *switches OpenFlow*.

4.1. Arquitetura

A arquitetura SD6WSN é uma arquitetura SDN típica, com uma camada de controle, composta por um controlador e agentes instalados nos nós, e uma camada de encaminhamento, composta pelos nós pertencentes à RSSF. A comunicação entre os controladores e os nós se dá através do plano de controle (provido pelo roteamento RPL) no qual trafegam as mensagens do protocolo SD6WSNP.

Como em outras arquiteturas SDN, o controlador é o responsável pelas funções básicas de controle da rede, como a descoberta e manutenção de topologia e comunicação na interface *Southbound*, ou seja, a função de troca de mensagens de controle com os nós, através do protocolo SD6WSNP. As aplicações que forem necessárias para otimizar as funções de rede no ambiente sem fio, tais como, o cálculo dos melhores caminhos dentro da RSSF, balanceamento de carga e controle de potência de transmissão, se comunicam com o controlador através da API *Northbound*.

Os nós SD6WSN são os mesmos *motes* utilizados em RSSF-6LoWPAN, aos quais foi adicionado um componente de *software* denominado “agente SD6WSN”, que controla a função de encaminhamento do plano de dados de acordo com as entradas presentes em sua tabela de fluxos. Destaca-se ainda a função de comunicação denominada “roteador de borda”, que estará

presente em nós específicos da rede, denominados 6LBRs, onde ocorre o roteamento dos pacotes entre a rede externa (IPv6) e o plano de controle da RSSF. No 6LBR também é incluído o mesmo agente SD6WSN existente nos demais nós, pois é nele que é realizado o encaminhamento dos pacotes de dados de aplicações entre a rede IPv6 externa e o plano de dados da SD6WSN. A Figura 4.1 ilustra a arquitetura SDWSN, com a divisão entre as funções de cada elemento.

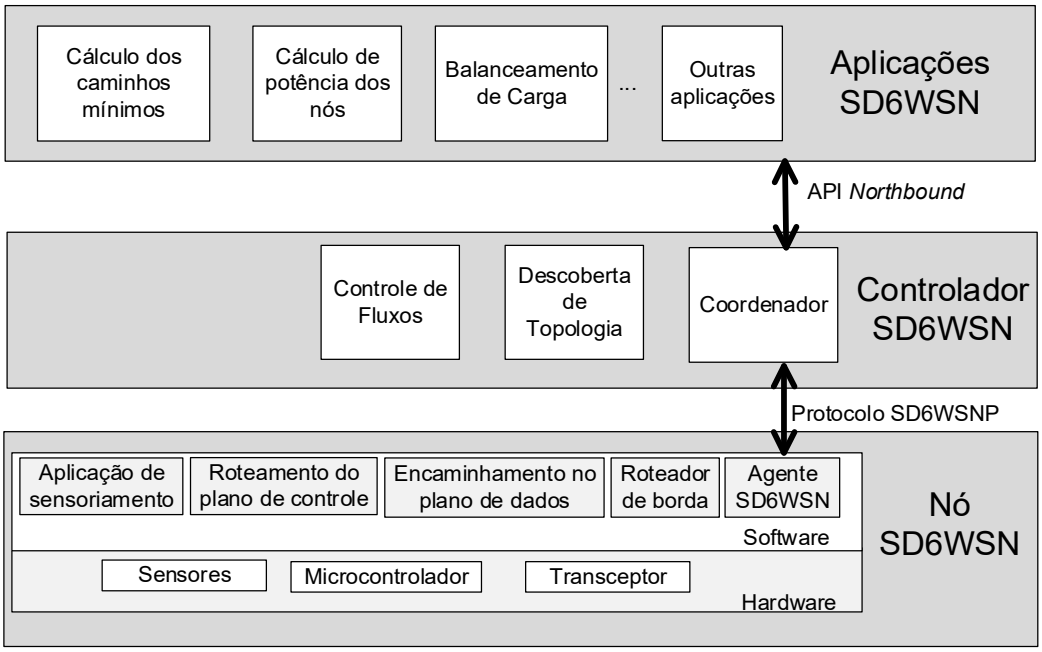

Figura 4.1 – Arquitetura SD6WSN

4.2. Comunicação entre os componentes da arquitetura

O protocolo de roteamento utilizado pelo plano de controle da arquitetura SD6WSN é o RPL em cima da camada 6LoWPAN. No plano de controle transitam as mensagens ICMPv6 responsáveis pela manutenção da rede RPL e também as mensagens do protocolo SD6WSNP. Caso existam nós tradicionais (não SD6WSN) na rede, a sua comunicação se dará pelos caminhos formados pelo RPL.

O ambiente RSSF está sujeito a maiores perdas de pacotes do que as redes cabeadas, o que desaconselha a utilização do protocolo TCP [SHE2011]. O TCP foi adotado pelo *OpenFlow* como protocolo de transporte para o plano de controle, já que o foco eram as redes cabeadas de alta velocidade. Uma alternativa para as limitações encontradas no TCP em RSSFs é a utilização do protocolo UDP, mas como este não possui o mecanismo de *handshaking* existente

no TCP, não se pode esperar, no nível de transporte, a garantia de entrega das mensagens. Para endereçar esta limitação, protocolos de camada de aplicação como o CoAP implementaram mensagens que solicitam confirmação de entrega, promovendo retransmissões das mesmas, caso as confirmações não retornem.

O protocolo SD6WSNP utiliza mensagens CoAP como base de sua comunicação entre o controlador e os nós SD6WSN. Essas mensagens tem o propósito de inserir informações nas tabelas de fluxos locais dos nós de ler os parâmetros de qualidade dos enlaces, localização e estado da bateria, medidos por eles.

Por meio de mensagens do protocolo SD6WSNP, o controlador envia instruções para o agente SD6WSN (que possui internamente um servidor CoAP), que por sua vez, interpreta as mensagens e executa ações, como a instalação de regras na tabela de fluxo. O agente SD6WSN também tem a função de interagir com os processos de mais baixo nível no sistema operacional (como o que implementa a pilha TCP/IP) para controlar o encaminhamento dos pacotes de acordo com a tabela de fluxos.

A Figura 4.2 apresenta a arquitetura de comunicação entre os componentes do sistema, sendo que o componente "nó" possui um número de instâncias igual ao número de nós presentes na RSSF.

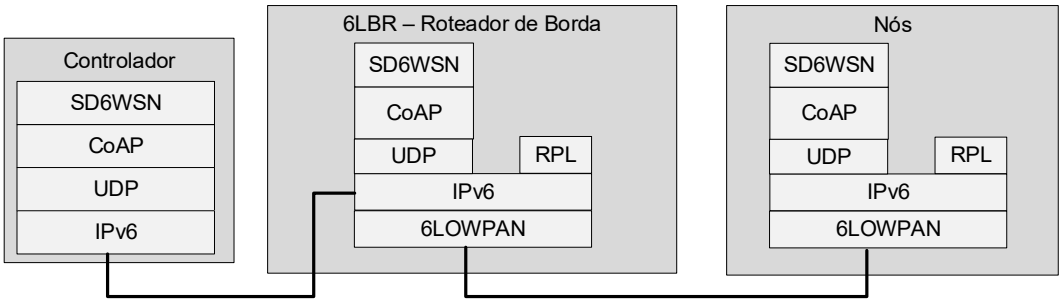

Figura 4.2 – Comunicação entre os componentes SD6WSN

A Figura 4.3 ilustra a comunicação feita pelo plano de controle, representada por linhas contínuas, onde trafegam as mensagens SD6WSNP. As linhas tracejadas ilustram um possível caminho para os pacotes de dados no plano de encaminhamento definido pela aplicação conectada ao controlador SD6WSN, diferente (para fins de exemplo) do caminho escolhido pelo RPL.

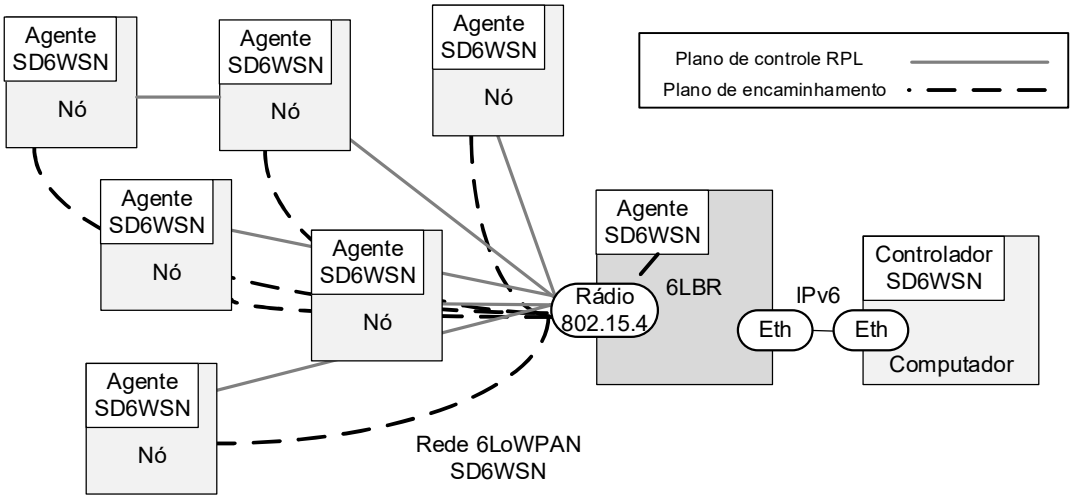

Figura 4.3 - Comunicação entre os nós e o controlador SD6WSN

4.3. Protocolo SD6WSNP

O protocolo SD6WSNP é o responsável pela comunicação entre o controlador e os nós SD6WSN no plano de controle. O formato da tabela de fluxo e de algumas mensagens do protocolo tiveram inspiração na especificação do protocolo *OpenFlow* 1.0 [ONF2010], porém adaptados para uso em ambientes de RSSFs.

4.3.1. Tabela de fluxos SD6WSN

As arquiteturas SDN baseadas em fluxos deixam a cargo de um controlador centralizado as tarefas de decisão de roteamento dentro de uma rede, o qual envia as informações necessárias para que os *switches* encaminhem corretamente os pacotes que ingressam em suas portas. Desta forma, o controlador estabelece regras para o encaminhamento dos pacotes, que são armazenadas na tabela de fluxo presente em cada *switch*. Somente pacotes que não possuam correspondência com as regras instaladas localmente é que são encaminhados para que o controlador decida qual a ação a ser tomada, o qual responde com uma nova regra a ser instalada no *switch* onde o pacote ingressou e nos demais que formam o caminho entre a origem e destino do pacote. Caso seja um pacote a ser descartado, uma regra com ação "*Drop*" é instalada apenas no *switch* de ingresso do pacote.

As regras de encaminhamento são armazenadas localmente nos *switche*s em estruturas de dados denominadas tabelas de fluxos, compostas basicamente por três campos:

- Campo de *match* ou de correspondência, onde são definidas as características do cabeçalho do pacote entrante que identificam um determinado fluxo;
- Campo de ação, onde é definida a ação a ser tomada com o pacote que foi identificado pela comparação do cabeçalho com o campo *match*;
- Contador de pacotes que corresponderam com o campo *match*.

Em redes de sensores, onde os nós tipicamente apresentam recursos de *hardware* limitados com necessidades de decisões de encaminhamento de pacotes mais simples, um número reduzido de campos de *match* pode ser adotado, contendo apenas os campos considerados relevantes, tais como *IPv6 Source Address, IPv6 Destination Address, TCP Source Port, TCP Destination Port, UDP Source Port e UDP Destination Port*. Além disso, para que seja possível a sumarização de rotas menos específicas em uma única entrada na tabela de fluxos, foram inseridos os campos IPv*6 Source Mask* e *IPv6 Destination Mask*. Campos como "*Ingress Port*", que não possuem função nas RSSFs, ou campos relativos às redes IPv4 (sem função em redes 6LoWPAN) foram excluídos da tabela de fluxos.

Por economia de espaço em memória, decidiu-se não incluir um campo para contagem de pacotes na tabela de fluxos dos nós SD6WSN. Caso futuramente esta informação se mostre necessária, ela poderá ser obtida indiretamente pela criação de uma nova mensagem do protocolo SD6WSNP.

A versão atual da tabela de fluxos na arquitetura SD6WSN é apresentada nas Tabelas 4.1 e 4.2, a primeira apresentando os campos de *match* e a segunda, as ações.

Tabela 4.1 – Lista de atributos para o campo *match* da tabela de fluxos SD6WSN

Campo *match*	**Comprimento (bits)**	**Descrição**
ipv6src	128	Endereço IPv6 de origem
srcmask	8	Máscara do endereço de origem (default /128)
ipv6dst	128	Endereço IPv6 de destino
dstmask	8	Máscara do endereço de destino (default /128)
srcport	16	Porta de transporte de origem
dstport	16	Porta de transporte de destino
ipproto	8	Protocolo IP (UDP(19), TCP(6) ou ICMPv6)

Tabela 4.2 – Lista de ações definidas na tabela de fluxos SD6WSN

Ações	Dados associados	Descrição
action	8 bits – código da ação a ser tomada no caso de *match*	0 – "Forward" – encaminha o pacote 1 – "Drop" – descarta o pacote 3 – "CPForward" – Encaminha via plano de controle
nhipaddr	128 bits - endereço IPv6 do nó de destino	Define o endereço IPv6 do próximo nó (*next-hop*) (parâmetro obrigatório da ação *Forward*)
rfpwr	8 bits – índice da tabela de potência do transmissor	Define uma nova potência de transmissão para o fluxo (parâmetro opcional da ação *Forward*)

Com a definição dos campos de *match* e de ações, cada entrada na tabela de fluxos tem a composição apresentada na Figura 4.4.

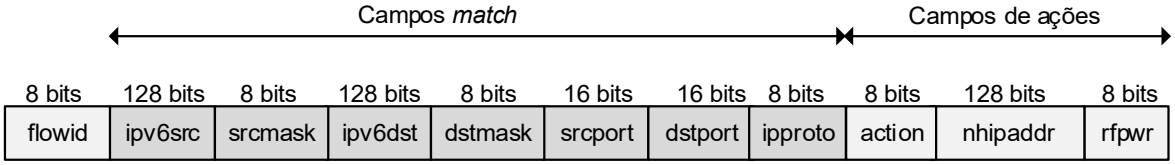

Figura 4.4 – Formato de uma entrada na tabela de fluxos

Cada entrada na tabela de fluxos possui um identificador único, numerado de 1 a 255, que corresponde ao campo "*flowid*" da tabela de fluxos, sendo que os índices mais baixos possuem prioridade maior caso possuam correspondência com o pacote que está sendo analisado, porém as entradas da tabela que possuam correspondências mais específicas com o pacote têm prioridade maior do que a definida pelo "*flowid*".

Algumas técnicas de agregação de endereços IP que empregam o campo *match* "*dstmask*" poderão ser usadas para diminuir o número de entradas nas tabelas, desde que o projeto de endereçamento IPv6 dos nós permita a agregação. Se este campo não for especificado, é assumida máscara /128, ou seja, somente o IPv6 indicado.

O número de campos *match* preenchidos em cada entrada da tabela de fluxo pode variar de zero a todos, sendo que os campos não informados são considerados como "coringas", ou seja, qualquer valor do cabeçalho do pacote é aceito. Caso nenhum dos campos *match* seja informado em uma entrada de fluxo, as ações correspondentes à essa entrada valerão para todos os pacotes que ingressarem no nó, a não ser que haja correspondência com alguma entrada mais específica.

4.3.2. Formatação das mensagens SD6WSNP

O protocolo SD6WSNP tem como base a troca de mensagens confirmadas do protocolo CoAP, que trafegam pelo plano de controle provido pelo protocolo RPL. As mensagens do protocolo SD6WSNP podem fluir nos dois sentidos, isto é, do controlador para um nó SD6WSN e vice-versa. Também estão previstas mensagens específicas entre o 6LBR e o controlador, para que o primeiro possa informar ao controlador o estado e a inclusão ou a exclusão de nós na RSSF.

O protocolo CoAP foi inicialmente pensado como um protocolo cliente-servidor, com as iniciativas de comunicações realizadas pelo cliente. Posteriormente foi estendido com a opção "*observe*" [RFC7641], ativada através de um *flag* nas mensagens do tipo GET enviada pelo cliente que instrui o servidor a enviar notificações ao mesmo sob determinadas condições, sem que haja uma nova requisição. No *framework* SD6WSN, o controlador assume o papel de cliente CoAP, e os nós assumem o papel de servidor, e a opção *observe* é utilizada nas mensagens SD6WSNP que utilizam o método GET, para que os nós enviem notificações assíncronas sempre que houver alguma mudança relevante em seu estado ou quando for recebido um pacote sem fluxo associado. Essa opção também é utilizada para que os nós 6LBR possam informar a identificação de novos nós na rede RPL.

O formato adotado para as mensagens de resposta foi o JSON (*Javascript Object Notation*) [RFC7159]. Este formato tem como vantagem a sua estrutura simples, com a delimitação dos campos formada por caracteres únicos, como chaves e colchetes, diminuindo assim o comprimento das mensagens de retorno.

As mensagens SD6WSNP podem ser classificadas de acordo com o processo ao qual estão associadas. O primeiro processo que é iniciado após a inclusão de um nó na RSSF é a descoberta e manutenção de topologia. As mensagens associadas com a descoberta e manutenção de topologia são *Node-mod* e *Info-get*. Após a inserção inicial do nó na topologia, as aplicações vinculadas ao controlador utilizarão o processo de controle de fluxos que utiliza a mensagem *Flow-mod* para a instalação e remoção de entradas nas tabelas de fluxo para que o novo nó execute o seu papel na RSSF. O controlador também enviará uma mensagem *Packet-in* ao novo nó, o instruindo a enviar-lhe uma notificação ao receber pacotes que não possuam correspondência na tabela de fluxos, no processo de processamento de pacotes sem fluxo associado. A Tabela 4.3 resume as mensagens do protocolo SD6WSNP, quais os elementos envolvidos e qual o processo a que estão relacionadas.

Tabela 4.3 – Mensagens do protocolo SD6WSNP

Mensagem	**Método CoAP**	***Observe***	**Processo**
Node-mod	GET	Obrigatório	Descoberta e manutenção da topologia
Info-get	GET	Obrigatório na primeira mensagem que solicita ao nó as informações sobre seus vizinhos, opcional nas demais mensagens	Descoberta e manutenção da topologia
Flow-mod	PUT	Não se aplica	Controle de fluxos
Packet-in	GET	Obrigatório	Processamento de pacotes sem fluxo associado

Node-mod, descrita na Tabela 4.4, é a primeira mensagem enviada pelo SD6WSNP, do controlador para o 6LBR, para registrar no 6LBR um pedido para que as entradas de novos nós na rede RPL sejam informadas. Desta forma, para cada novo nó identificado pelo 6LBR (pelo recebimento de uma mensagem RPL do tipo DAO), uma notificação *observe* informa o controlador a existência desse nó.

Tabela 4.4 – Descrição das mensagens *Node-mod*

Descrição	**Requisição SD6WSNP**	**Retorno JSON**
Novo nó no 6LBR (*observe*=obrigatório)	/sd6wsn/node-mod	`Novo nó: {"nodeadd":" <IPv6 do novo nó>"}` `Remoção: {"nodedel":" <IPv6 do nó removido>"}`

Exemplo de requisição do controlador para o 6LBR:

`GET coap://[2001:0DB8::1]/sd6wsn/info-get/node-mod (observe)`

Exemplo de notificação quando do registro de um novo nó no 6LBR:

`{"nodeadd":"2001:0DB8::10"}`

Exemplo da remoção do registro de um nó no 6LBR:

`{"nodedel":"2001:0DB8::11"}`

Info-get, que tem suas opções descritas na Tabela 4.5, é utilizada sempre que o controlador precisa de alguma informação sobre o nó, mas é utilizada principalmente pelo processo de descoberta e manutenção de topologia, onde uma requisição com a opção *observe* é utilizada para que o controlador receba alterações nas métricas de qualidade dos enlaces sempre que houver alguma alteração significativa em um ou mais enlaces.

Tabela 4.5 – Descrição das mensagens *Info-get*

Descrição	Requisição SD6WSNP	Retorno JSON
ETX dos nós vizinhos (*observe* obrigatório na primeira mensagem para o nó e opcional nas demais mensagens)	/sd6wsn/info-get/nbr-etx	{"node":"n<IPv6 do nó>", "nbr":{"n<IPv6 do vizinho 1>":<ETX do vizinho 1>,"n<IPv6 do vizinho 2>":<ETX do vizinho 2>,"n<IPv6 do vizinho n>":<ETX do vizinho n>}}
RSSI (*observe* opcional)	/sd6wsn/info-get/rssi?nbr=<IPv6 do vizinho>	{"node":"n<IPv6 do nó>","n<IPv6 do vizinho 1>:<RSSI do vizinho>}
Estado da bateria (*observe* opcional)	/sd6wsn/info-get/battery	{"node":"n<IPv6 do nó>","vbat”:<tensão em mV>}
Localização (*observe* opcional)	/sd6wsn/info-get/location	{"node":"n<IPv6 do nó>","lat":<latitude>, "lon":<longitude>, "elev":<elevação>}

Duas métricas de qualidade de enlaces estão disponíveis por requisições *Info-get*, o RSSI (*Received Signal Strength Indicator*) que indica o nível de sinal de RF do último quadro recebido, e o ETX, que mede o número de retransmissões que foram necessárias para o envio de um pacote. A métrica para medida adotada neste trabalho é o ETX, com o mesmo tratamento que o Contiki adota em sua implementação do RPL, com a aplicação de um filtro EWMA (*Exponentially Weighted Moving Average*), que suaviza as variações do ETX medido utilizando a série histórica para compor o valor atual, minimizando assim o efeito que variações ocasionais poderiam ter no cálculo de qualidade do enlace.

Algumas implementações do RPL, como na do sistema operacional Contiki, utilizam a técnica de suavização exponencial que faz uso dos valores passados de ETX para o cálculo do valor corrente, evitando assim que um problema momentâneo de transmissão cause uma mudança abruta do seu valor. Com a possibilidade do valor ETX ser um número decimal, a [RFC6551]

definiu que o para o seu uso como métrica de qualidade, ele deve ser multiplicado por 128 e transformado em um valor inteiro de 16 bits. No *framework* SD6WSN utilizamos o ETX com essa modificação, e para a simplificação do texto usaremos apenas a sigla ETX, sem mencionar a multiplicação por 128.

Um exemplo de uma requisição *Info-get* é a feita pelo controlador ao nó de endereço IPv6 2001:0DB8::8, para obtenção dos valores de ETX de seus vizinhos:

```
GET coap://[2001:0DB8::8]/sd6wsn/info-get/nbr-etx
```

O retorno para esta requisição vem em formato JSON, como no exemplo abaixo:

```
{
      {"node":"n8","nbr":{"n5":450,"n9":362,"n7":285}}
}
```

O retorno para esta requisição mostra que o nó com endereço IPv6 de final ::8 tem três vizinhos (nós com endereços finais ::5, ::9 e ::7) e com ETX igual a 450, 368 e 285, respectivamente.

Packet-in, descrita na Tabela 4.6, é enviada a um nó logo após a sua descoberta pelo controlador, para registrar no mesmo uma instrução para que informações sobre os pacotes do plano de dados que não possuam uma correspondência com nenhuma entrada na tabela de fluxos sejam enviadas para o controlador.

Tabela 4.6 – Descrição das mensagens *Packet-in*

Descrição	**Requisição SD6WSNP**	**Retorno JSON**
Pacote recebido sem fluxo associado (*observe* obrigatório)	/sd6wsn/packet-in	`{"node":"n<IPv6 do nó de entrada>", {"packetin" :{"ipv6src":"<IPv6 de origem do pacote>","ipv6dst":"<IPv6 de destino do pacote>","srcport":<porta de origem do pacote>,"dstport":<porta de destino do pacote>,"ipproto":<protocolo de transporte do pacote>}}`

Um exemplo de requisição *Packet-in* com a opção *observe* ativada é:

`GET coap://[2001:0DB8::10]/sd6wsn/packet-in` (com a opção *observe*)

Uma notificação enviada pelo agente SD6WSN ao controlador quando da entrada de um pacote sem correspondência na tabela de fluxos seria como a abaixo, com os dados do cabeçalho do pacote que ingressou no nó e que precisa ser tratado:

```
{
      {"node":"n10",{"packetin":{"ipv6src":"2001:0DB8::110","ipv6dst":
      "2001:0DB8::56","srcport":3210,"dstport":80,"ipproto":19}}
}
```

A mensagem *Flow-mod* é a única que utiliza o método PUT e é enviada pelo controlador sempre que uma das aplicações requeira que a tabela de fluxos de um ou mais nós SD6WSN seja alterada. A Tabela 4.7 apresenta o formato das mensagens do SD6WSNP para a inserção e modificação de dados nas tabelas de fluxos dos nós.

Tabela 4.7 – Mensagens SD6WSNP para a manutenção da tabela de fluxos

Descrição	Requisição SD6WSNP
Insere entrada na tabela de fluxos	/sd6wsn/flow-mod?operation=insert?flowid=n&ipv6src=<endereço de origem>&ipv6dst=<endereço de destino>&action=<ação a ser tomada> &nhipaddr=<endereço IPv6 de próximo salto>&txpwr=<potência>
Apaga a entrada de fluxo "n" da tabela	/sd6wsn/flow-mod?operation=delete&flowid=n

Um exemplo de requisição *Flow-mod* feita pelo controlador para a inserção de uma entrada do tipo "*Forward"* (descrita na Seção 4.4.3) na segunda posição da tabela de fluxos do nó 2001:0DB8::10 seria:

```
      PUT coap://[2001:0DB8::10]/sd6wsn/flow-mod?operation=insert&flowid=2
&ipv6dst=2001:0DB8::10:20&action=0&nhipaddr=fe80::10:10&txpwr=3
```

Neste exemplo, a entrada da tabela de fluxos determina que todos os pacotes que tenham como destino o IPv6 2001:0DB8::10:20 sejam encaminhados para o nó que tem o endereço IPv6 link-local fe80::10:10 e com potência de transmissão definida pelo índice "3" da tabela de potências interna ao *mote* (que depende do modelo do mesmo), conforme explicado no capítulo 5.

4.4. Controlador SD6WSN

Na arquitetura SD6WSN, o controlador é constituído por dois processos, um que tem a função de descobrir e realizar a manutenção da topologia da RSSF e outro, "coordenador" que utiliza a interface *Southbound* para a comunicação com os nós e *Northbound* para a comunicação com as aplicações que farão uso da infraestrutura.

O processo de descoberta de topologia é acionado por uma mensagem do 6LBR para o processo coordenador sempre que houver a identificação de um novo nó na rede RPL.

4.4.1. Coordenador SD6WSN

O coordenador é responsável por enviar e receber as mensagens do SD6WSNP, e por interagir com a função de Descoberta de Topologia do controlador e com as aplicações, através de mensagens REST e JSON, conforme mostrado na Figura 4.5. As rotinas de controle da RSSF são escritas de forma a sincronizar as ações de acordo com os diagramas de sequência que serão apresentados nas próximas seções, com os dados presentes em um banco de dados interno, onde os estados dos componentes da RSSF são armazenados.

Para a função de comunicação através de mensagens do SD6WSNP, o coordenador emprega uma biblioteca de cliente CoAP na linguagem escolhida para a escrita do controlador (C, Node.js, Python e etc.) que implementa as mensagens de acordo com a [RFC7252], bem como a opção *observe.* O coordenador é compilado para o sistema operacional escolhido para hospedar o controlador. Na implementação atual, o controlador foi compilado para o sistema operacional Linux, mas pode ser portado para outros sistemas operacionais.

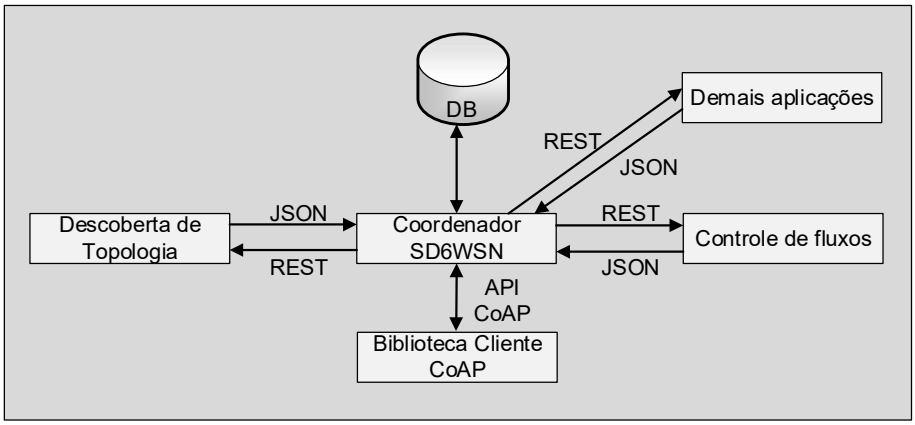

Figura 4.5 – Diagrama de interação do coordenador com os demais processos do controlador

A utilização de APIs do tipo REST e JSON para a comunicação permite que as aplicações estejam distribuídas em diversos sistemas, com o intuito de prover a capacidade de processamento e redundância necessárias para os mais diversos cenários de aplicação.

4.4.2. Descoberta e manutenção de topologia

Para manter sua visão da topologia da rede, o controlador precisa identificar os nós ativos da rede, mas também os vizinhos que cada um deles enxerga, incluindo a qualidade dos enlaces que os conectam, indicados pela métrica ETX. O protocolo SD6WSNP utiliza informações do protocolo do plano de controle (RPL) no processo de descoberta de topologia. Esse processo se inicia logo após o início da formação do DODAG pelo RPL e o consequente registro das rotas de acesso aos nós no 6LBR por meio de mensagens RPL DAO.

Node-mod é a primeira mensagem enviada pelo controlador para o nó 6LBR, com a opção *observe* ativada. Depois disso, quando um novo nó RPL é registrado ou se um nó deixa de aparecer na tabela de rotas, o 6LBR retorna uma notificação para o controlador, informando a inclusão ou exclusão do nó. No caso de inclusão, o controlador verifica se o nó registrado no 6LBR faz parte de uma tabela pré-carregada dos endereços IPv6 dos nós autorizados a participar da rede SD6WSN. Também podem estar configuradas nessa tabela as coordenadas geográficas e as restrições que alguns nós possam ter, como energia no caso de nós alimentados por bateria. Se o novo nó for um nó autorizado, o controlador primeiramente envia a ele as requisições *Info-get* (apresentadas na Tabela 4.5) que tem como retorno as suas informações físicas, como a localização geográfica e o estado da bateria. Estas informações são de caráter opcional e podem ser acrescidas da opção *observe*, caso a seja necessário o acompanhamento da variação desses parâmetros.

Em seguida, é enviada uma requisição *Info-get/nbr-etx* ao novo nó para a coleta da informação dos seus vizinhos visíveis e da qualidade (ETX) do enlace entre ele e cada um dos seus vizinhos. Essa mensagem é enviada obrigatoriamente com a opção *observe* ativada, de modo que essa informação seja enviada periodicamente pelo nó SD6WSN ao controlador. Também foi prevista uma requisição opcional *Info-get/*rssi (ver Tabela 4.5) que tem como retorno o RSSI do nó vizinho (obtido via *Info-get/etx*), informado na requisição.

A Figura 4.6 apresenta o diagrama de sequência de descoberta de topologia.

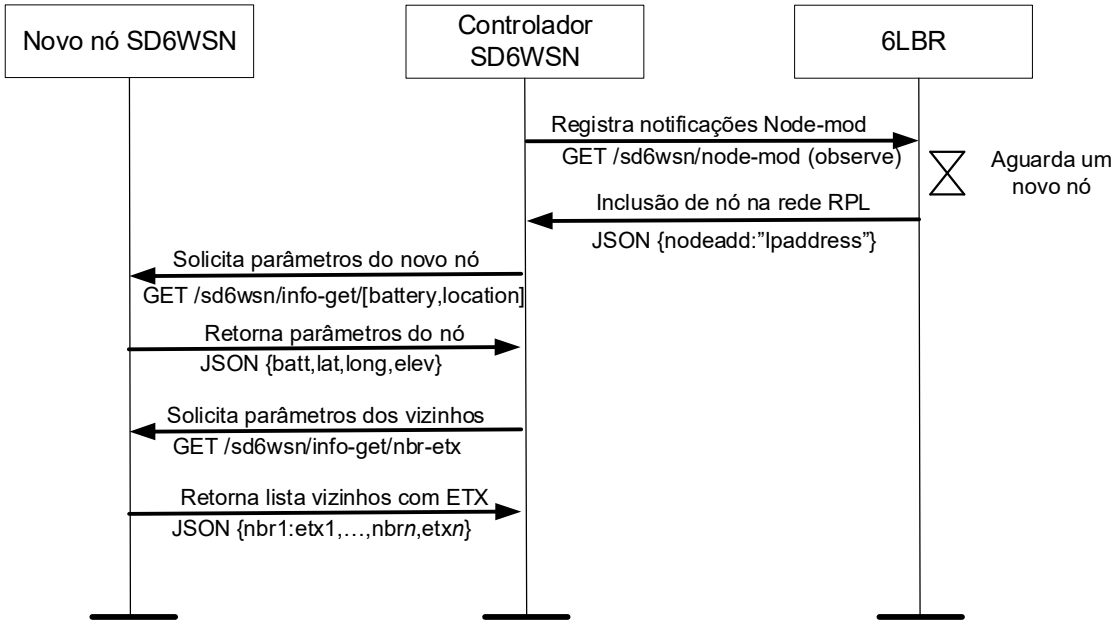

Figura 4.6 – Descoberta de topologia

Quando da exclusão de um nó por seu desaparecimento na tabela de roteamento do 6LBR, as aplicações são avisadas da mudança de topologia pelo controlador e decidem se há a necessidade de um recálculo dos caminhos dos fluxos dentro da RSSF.

Além da informação sobre a mudança do número de nós, o controlador deve manter a informação dos vizinhos e da qualidade dos respectivos enlaces para cada nó. Para isso, conforme dito anteriormente, a mensagem *Info-get/nbr-etx* é enviada para os nós com a opção *observe* ativada. Cada nó verifica periodicamente a presença dos seus vizinhos e a qualidade dos respectivos enlaces, e quando há uma alteração significativa na qualidade de um ou mais enlaces, ou modificações de vizinhança, ele envia ao controlador uma notificação de resposta à mensagem *Info-get* correspondente. Ao receber a informação, o controlador informa as mudanças para as aplicações, que realizam o recálculo dos fluxos existentes e informam ao controlador quais as modificações que são necessárias. A Figura 4.7 apresenta o diagrama de sequência das mensagens de manutenção de topologia.

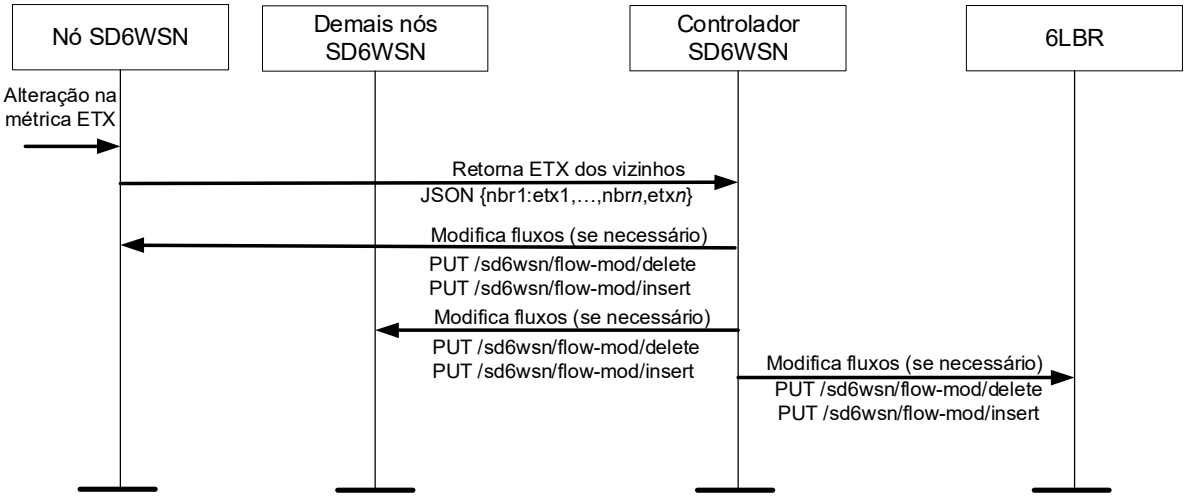

Figura 4.7 – Manutenção de topologia

Em algumas situações, como em uma reinicialização de um nó, por exemplo, é possível que um registro *observe* seja perdido. Para contornar este tipo de situação, o controlador inicia um temporizador sempre que uma mensagem *Info-get/nbr-etx* é enviada para um nó, que é reiniciado sempre que uma resposta desse comando chega ao controlador. Caso o temporizador ultrapasse o limite definido, uma nova mensagem *Info-get* é enviada para o nó em questão (ver Figura 4.8).

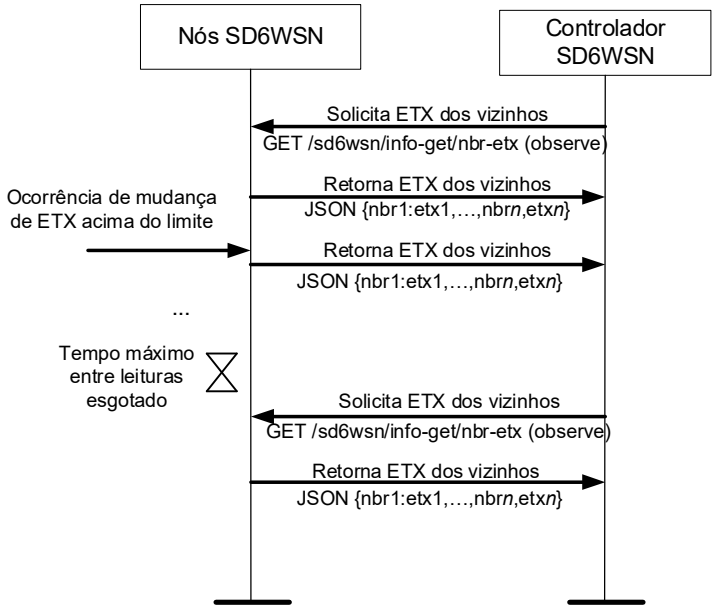

Figura 4.8 – Atualização das informações de um nó previamente existente.

As aplicações podem necessitar, a qualquer momento, a obtenção de parâmetros específicos dos nós, como nível de energia da bateria e a potência de transmissão. Se a aplicação não necessitar receber notificações de alteração nos parâmetros, ou se forem imutáveis, como no caso de nós sem bateria, as requisições são realizadas por demanda pelo controlador, quando houver necessidade de atualização dos parâmetros. Por este motivo, os parâmetros de coordenadas geográficas e nível de bateria são lidos por requisições GET sem a opção *observe* associada. A Figura 4.9 apresenta o diagrama de sequência da leitura dos parâmetros mencionados.

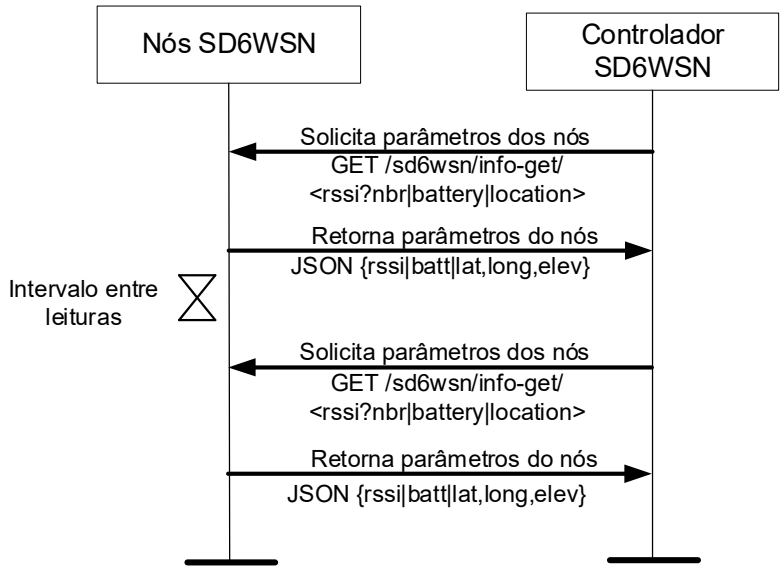

Figura 4.9 – Leitura de parâmetros específicos

4.4.3. Processo de controle de fluxos

O controle de fluxos é realizado por algoritmos executados por aplicações conectadas à interface *Northbound* do controlador, cujo resultado é a instalação de entradas nas tabelas de fluxos dos nós. A instalação ou remoção de entradas na tabela de fluxos nos nós são efetuadas por mensagens SD6WSNP *Flow-mod*. As operações possíveis são duas, de inserção (*insert*) ou remoção (*delete*) de uma entrada. Caso uma entrada já exista, ela será sobrescrita. A instalação de um fluxo pode decorrer de uma decisão de uma aplicação como consequência da recepção pelo controlador de mensagem "*Packet-in*", enviada por algum nó, quando esse recebe uma mensagem no plano de dados, para a qual não existe uma entrada na tabela de fluxos que determine o procedimento de encaminhamento.

Para fins de comparação, o protocolo SDN *OpenFlow*, que tem como foco as redes cabeadas formadas por *switches*, possui em algumas ações o encaminhamento dos pacotes de um fluxo para uma determinada porta de um *switch*. Em nós RSSF isto não ocorre, porque os pacotes são encaminhados para outros nós por radiofrequência, existindo apenas uma entrada e uma saída em cada nó. Neste caso, a principal ação para as RSSFs seria a modificação do endereço IPv6 de próximo salto. As seguintes ações foram definidas para o protocolo SD6WSNP:

- *Forward*, ação de encaminhamento dos pacotes identificada por uma entrada com valor "0" no campo *action* na entrada da tabela de fluxos, a qual define o valor do endereço IPv6 do próximo salto pelo parâmetro obrigatório *nhipaddr* e com a potência de transmissão definida pelo parâmetro opcional *txpwr*;
- *Drop,* que é a ação de descarte dos pacotes de um determinado fluxo, definido pelo valor "1" no campo *action* na entrada da tabela de fluxos. Os parâmetros *nhipaddr* e *txpwr* são ignorados;
- *CPForward* é uma ação que encaminha os pacotes pertencentes ao fluxo para serem roteados pelo plano de controle. É definido por um valor "2" no campo *action* e os parâmetros *nhipaddr* e *txpwr* são ignorados.

As entradas na tabela de fluxos com a ação "*Forward*" deverão ter obrigatoriamente o endereço IPv6 de nó de destino dos pacotes. Caso exista mais de uma entrada que possua correspondência para um mesmo fluxo, a mais específica, ou seja, a que contenha informações mais completas nos campos de correspondência será considerada. Se todas as entradas possuírem informações equivalentes, somente a primeira (mais prioritária) é considerada.

As entradas "*Forward*" possuem um parâmetro opcional para a determinação da potência de transmissão do rádio, válido somente para a própria entrada. A potência determinada é somente válida para os pacotes pertencentes ao fluxo, e ela é retomada para o valor *default* logo após a transmissão de cada um desses pacotes. Os pacotes do plano de controle são sempre transmitidos com a potência *default* do rádio.

4.4.4. Inclusão de novos fluxos a partir de pacotes recebidos pelos nós

Quando um nó recebe um pacote para encaminhamento externo, e que não seja uma mensagem do plano de controle (ver Figura 4.10), alguns de seus campos de cabeçalho IP são

analisados segundo as regras de correspondência da tabela de fluxos, verificando junto à tabela de fluxos se já existe uma entrada definida para um pacote de dados com tais características. Caso exista alguma entrada correspondente, isto é, se ocorrer um *matching*, ele executa a ação associada. Esta ação pode ser o descarte do pacote ou seu encaminhamento para outro nó.

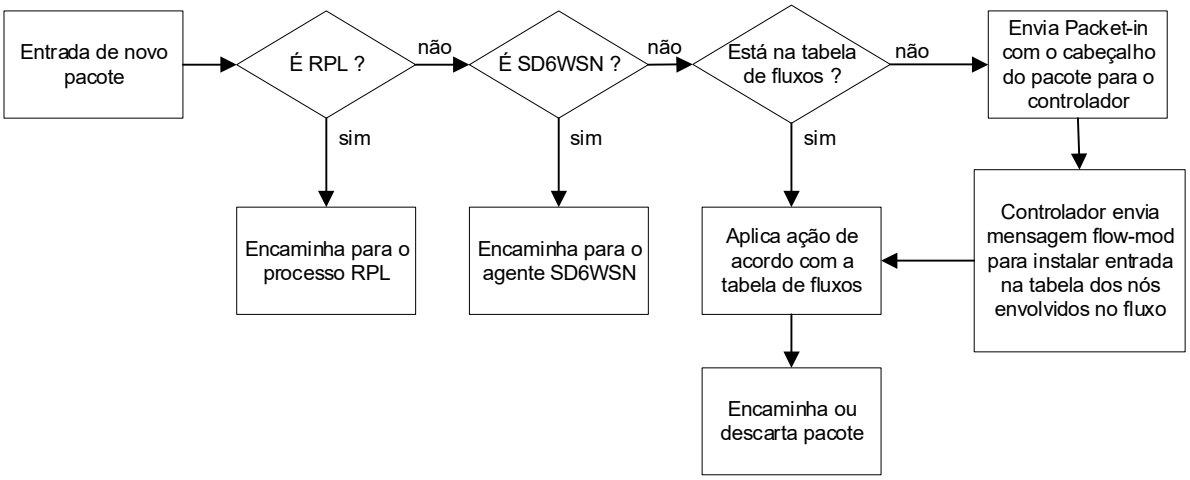

Figura 4.10 – Fluxograma de tratamento de entrada de novos pacotes em um nó

Para o caso de recepção de pacote de fluxos ainda não existente na tabela, o controlador envia para cada nó uma mensagem *Packet-in* com a opção *observe* ativa. Sempre que não ocorrer correspondência entre o cabeçalho do pacote entrante com alguma entrada da tabela de fluxos, o nó retorna ao controlador uma resposta à mensagem *Packet-in* através de uma mensagem JSON contendo o cabeçalho do pacote recebido. Ao receber essa resposta, o controlador determina a "rota" a ser seguida pelos pacotes do novo fluxo e envia aos nós envolvidos mensagens *Flow-mod*. Essas mensagens acrescentam entradas nas tabelas de fluxos dos nós pertencentes à rota com o padrão de correspondência e com a ação que deve ser tomada para este pacote e para os demais pacotes com características semelhantes, normalmente integrantes do mesmo fluxo. A Figura 4.11 apresenta o diagrama de sequência do processo de entrada de um pacote sem um fluxo associado.

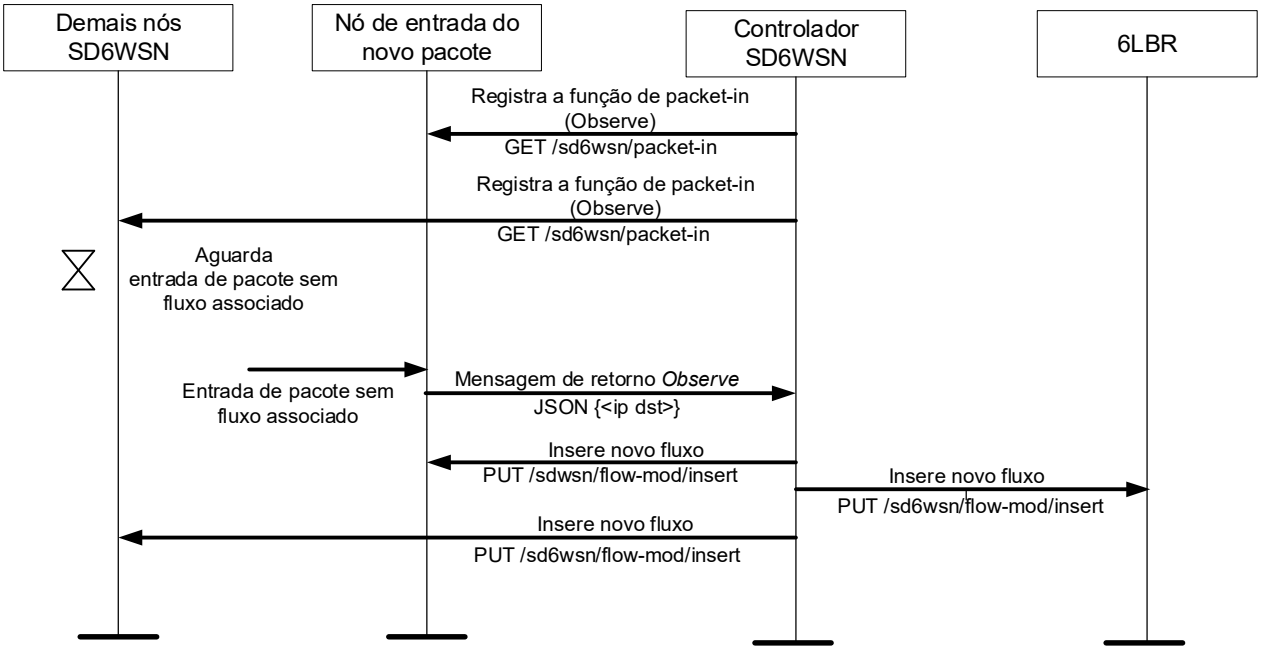

Figura 4.11 – Diagrama de sequência de mensagens *Packet-in*

4.5. Nós SD6WSN

Esta Seção apresenta a arquitetura dos nós SD6WSN, composta pelo agente SD6WSN e pelas funções de comunicação, que englobam o roteamento do plano de controle e o encaminhamento de pacotes no plano de dados. Também é apresentado o nó denominado 6LBR, que além das demais funções de um nó SD6WSN, possui a função especial de roteamento de borda.

4.5.1. Agente SD6WSN

O agente SD6WSN é um componente de *software* que é instalado nos nós e no 6LBR. Esse agente tem a função de receber as mensagens do protocolo SD6WSNP e interagir com o processo de encaminhamento de pacotes do sistema operacional. É baseado em um servidor CoAP, para o qual foram desenvolvidos recursos mapeados nas mensagens do protocolo SD6WSN.

As principais funções do agente são:

- Enviar ao controlador as informações da rede, como os vizinhos que são visíveis e as métricas de qualidade dos enlaces com eles;
- Gerenciar as entradas na tabela de fluxos interna ao nó;

- Inspecionar os cabeçalhos dos pacotes de saída e enviar mensagens *Packet-in* ao controlador caso não estejam na tabela de fluxos;
- Interagir com o processo de encaminhamento de pacotes do sistema operacional para encaminhar os pacotes que possuam correspondência na tabela de fluxos de acordo com o especificado na mesma;
- Interagir com outros processos do sistema operacional dos nós para alterar parâmetros de transmissão, como o canal e potência de RF.

O diagrama da Figura 4.12 mostra a integração do agente SD6WSN com as demais camadas de comunicação do sistema operacional. Todos os nós executam um sistema operacional embarcado como o Contiki [DUN2004], que possui nativamente os componentes PHY *Driver*, MAC, 6LoWPAN, IPv6 e RPL.

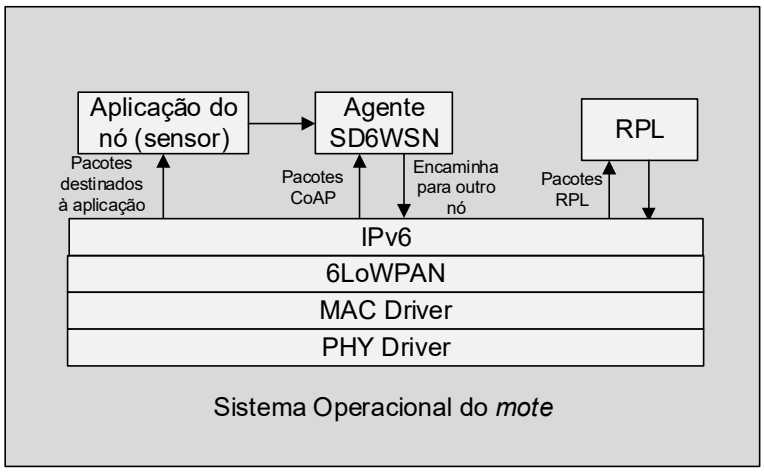

Figura 4.12 - Integração do agente SD6WSN com o sistema operacional do nó

4.5.2. Funções de comunicação

Todos os nós SD6WSN devem incluir a função de encaminhamento do plano de dados, a função de roteamento do plano de controle, e pelo menos um dos nós deve incluir a função de roteador de borda.

O processo de encaminhamento da arquitetura de *software* original do sistema operacional foi modificado para contemplar as funções de comunicação dos nós SD6WSN, conforme apresentado na Figura 4.13. Dois tipos de pacotes podem iniciar um novo fluxo: pacotes gerados internamente ao *mote* pela aplicação de sensoriamento local ou por pacotes

oriundos de outros nós da rede RSSF. Em ambos os casos, o pacote será tratado pela camada de rede, que pode determinar o descarte, o encaminhamento interno ou o encaminhamento externo. O encaminhamento interno é dirigido para a camada de transporte, que verifica se é ou não um pacote do protocolo SD6WSNP, o encaminhando para a porta UDP adequada. Se tratar-se de um pacote com porta de destino diferente da utilizada pelo SD6WSNP, o pacote é direcionado para a aplicação local.

O encaminhamento externo pode se dar através da função padrão existente no sistema operacional quando for um pacote do plano de controle, que deve ser tratado pelo RPL. Se não for um pacote do plano de controle, necessariamente será do plano de dados, e deverá passar pelo controle de fluxos presente no agente SD6WSN antes de ser encaminhado para a interface sem fio.

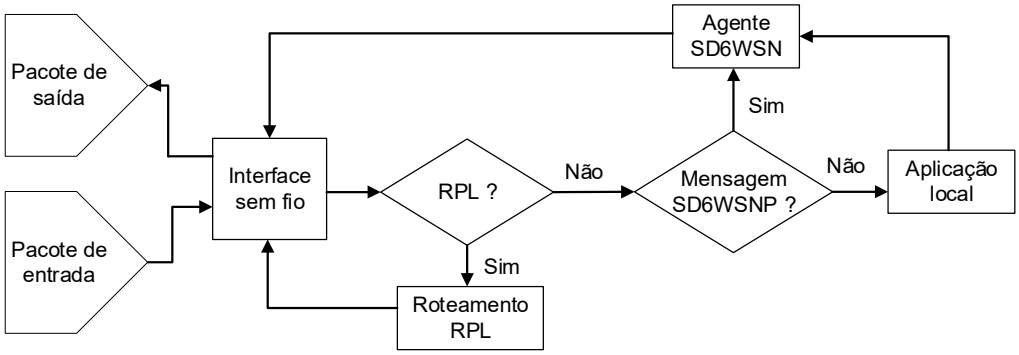

Figura 4.13 - Processo de encaminhamento de pacotes

A função de roteamento do plano de controle é realizada pelo RPL, seguindo as suas especificações, descritas no Capítulo 2.

A função de encaminhamento do plano de dados é o que transforma um nó SD6WSN em um "switch" SDN. Essa função é que realiza o desvio dos pacotes de dados das aplicações dos dados do plano de controle e os encaminha para o nó de próximo salto de acordo com o endereço IPv6 definido em um campo de ação na tabela de fluxos. Caso o parâmetro de definição de endereço de próximo salto *nhipaddr* seja nulo, o pacote será descartado, mas se não houver correspondência ao cabeçalho do pacote nos campos *match* nas entradas da tabela de fluxos, uma notificação *Packet-in* será gerada, instruindo a aplicação a tomar uma decisão de qual ação tomar, seguida de uma solicitação para que controlador instale um fluxo no nó de entrada desse pacote e nos demais nós envolvidos no fluxo.

A função “roteador de borda” da arquitetura SD6WSN é implementada através de uma extensão no roteador 6LBR. A função original do roteador 6LBR (nas redes 6LoWPAN/RPL) é interligar as redes IPv6 e 6LoWPAN. Na arquitetura SD6WSN o roteador 6LBR é estendido através do agente SD6WSN, como visto na Figura 4.14, que recebe do controlador as entradas da tabela de fluxo com instruções de encaminhamento de pacotes no plano de dados.

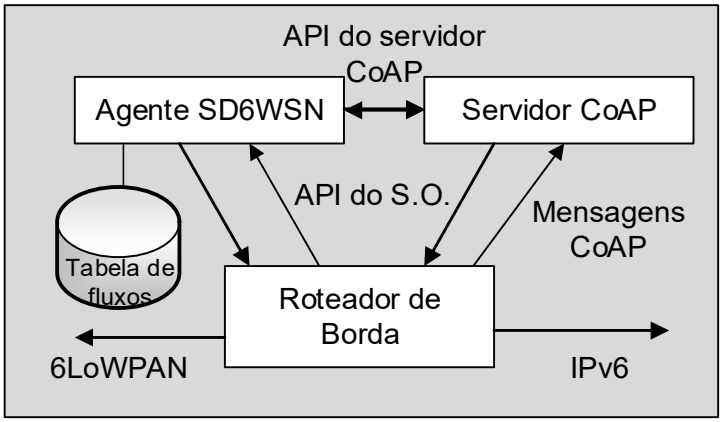

Figura 4.14 – Integração da função “roteador de borda”

O roteador 6LBR com extensão SD6WSN passa a ter duas funções, uma de realizar o roteamento entre as redes IPv6 e 6LoWPAN, e uma outra de identificar os pacotes pertencentes aos fluxos definidos pelo controlador SD6WSN e tratá-los da forma adequada. Caso os pacotes recebidos pelo 6LBR sejam do protocolo SD6WSNP, tanto vindos do controlador quanto vindo dos nós, este tráfego é identificado como tal e o 6LBR faz o roteamento RPL sem o desvio para o agente, a não ser que seja um pacote destinado ao agente do próprio 6LBR. Porém, quando o 6LBR recebe pacotes que não sejam do plano de controle e destinados à sub-rede definida para a RSSF, o agente SD6WSN controla o encaminhamento dos pacotes obedecendo às informações da tabela de fluxos.

Outra responsabilidade do 6LBR com extensão SD6WSN é alertar o controlador quando um nó é detectado ou deixa de existir na RSSF. Isto é realizado por uma notificação *Node-mod* do protocolo SD6WSN, que é enviada quando uma nova rota com destino à RSSF é inserida ou retirada pelo RPL da tabela de rotas do 6LBR.

4.6. Aplicações SD6WSN

Esta seção apresenta algumas aplicações que podem ser desenvolvidas para modelar o fluxo de pacotes dentro da RSSF SDN de acordo com objetivos específicos, tais como prover uma comunicação otimizada entre os nós, com balanceamento de carga entre os enlaces ou permitir a priorização de alguns tipos de dados de aplicações. As aplicações interagem com o controlador através de APIs *Northbound* disponibilizadas pelo coordenador, como mostrado na Figura 4.1.

4.6.1. Cálculo dos caminhos mínimos entre os nós e o 6LBR

A primeira aplicação idealizada para esta arquitetura é a determinação dos melhores caminhos dentro da RSSF entre cada nó e o 6LBR (e vice-versa) para qualquer tipo de dado de aplicação que trafegue pela rede. Esta aplicação tem o objetivo de prover a conectividade no plano de dados para qualquer tipo de dado de aplicação entre os nós e o 6LBR e consequentemente, à rede IPv6 que está conectada a ele. Em RSSFs, este é o tráfego mais comum, onde os dados de sensoriamento são direcionados para um servidor conectado fora da RSSF-6LoWPAN.

Conforme o controlador obtém as informações da topologia, a aplicação de caminhos mínimos calcula o melhor caminho entre cada nó e o 6LBR utilizando as informações de quais são os vizinhos de cada nó, quais são os vizinhos comuns entre eles e quais são os valores de qualidade de cada enlace. O algoritmo escolhido para aplicação foi o algoritmo de Dijkstra [SED2011], utilizando os nós e seus vizinhos como vértices e os valores de ETX como sendo os custos entre eles, representados nas arestas. A Figura 4.15 exemplifica o resultado do cálculo de caminhos mínimos para um conjunto de nós com os custos indicados nas arestas. Os caminhos preferenciais calculados pelo algoritmo são indicados pelas setas sólidas, enquanto que possíveis enlaces alternativos (não escolhidos) são indicados por linhas pontilhadas. Os caminhos inversos, entre o nó raiz (representado pela letra “R”) e os demais nós, não foram representados na Figura 4.15, mas utilizam um cálculo semelhante, já que a informação do ETX provê os custos dos enlaces em ambos os sentidos.

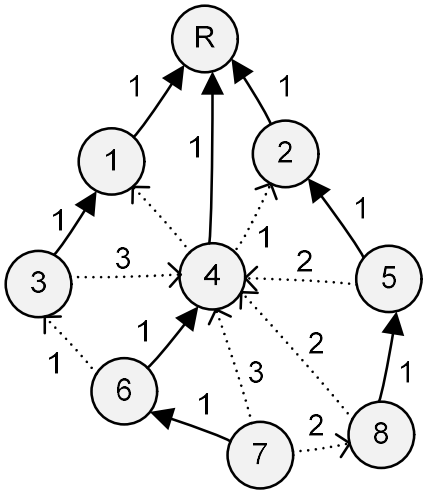

Figura 4.15 – Exemplo de uma árvore de caminhos mínimos

O algoritmo é executado sempre que há uma alteração nos seguintes fatores da RSSF:

- Inclusão ou exclusão de um nó;
- Alteração das restrições de um nó, como baixa autonomia de bateria;
- Alteração na qualidade (indicada pela variação do ETX) dos enlaces acima de um determinado percentual.

Os fluxos de caminhos mínimos começam a ser instalados logo após a entrada do primeiro nó na RSSF e a leitura de seus parâmetros. Após o recebimento da mensagem e verificação se o nó existe na tabela interna, o controlador o insere na estrutura de dados que representa a topologia corrente e calcula (ou recalcula) o caminho com menor custo ao 6LBR e o caminho inverso.

Após o cálculo de caminhos mínimos para cada nó, o controlador envia as mensagens *Flow-mod* para todos os nós envolvidos em cada fluxo para a instalação de entradas nas tabelas de fluxo de todos os nós. Se o cálculo (ou recálculo) da árvore de caminho mínimo alterar os caminhos mínimos e este cálculo envolver outros nós, estes também receberão atualizações nas entradas de suas respectivas tabelas de fluxo.

A Figura 4.16 apresenta o diagrama de sequência e instalação dos fluxos pertencentes aos caminhos mínimos logo após o registro de um novo nó na RSSF.

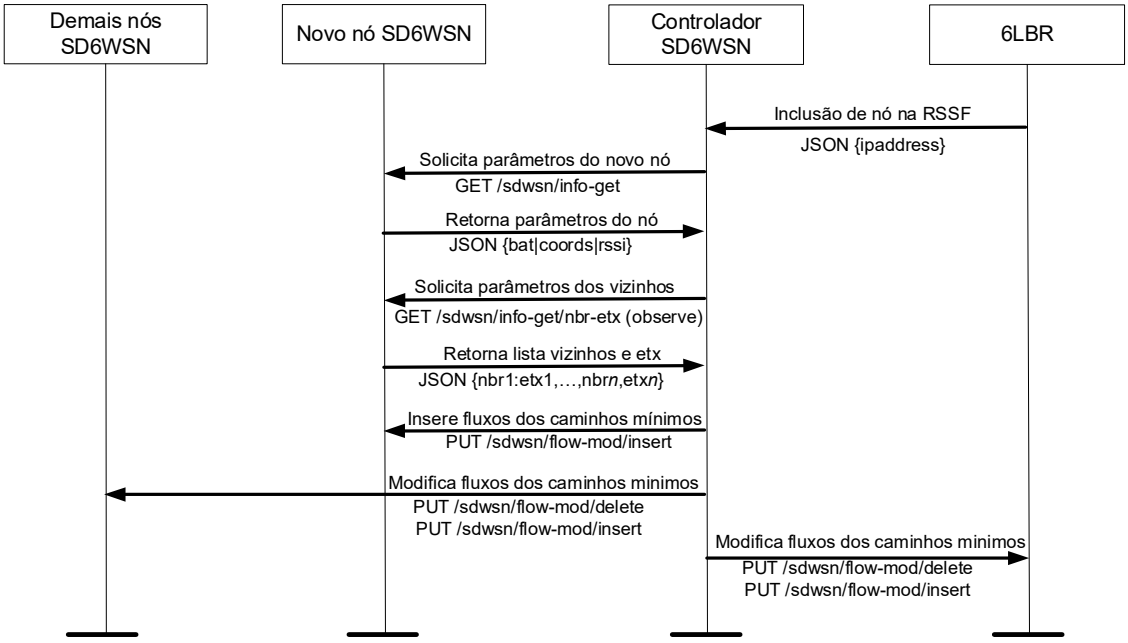

Figura 4.16 – Instalação dos fluxos pertencentes aos caminhos mínimos

Diferentemente do RPL, que assume que a qualidade dos enlaces entre dois nós é a mesma em ambos os sentidos, esta aplicação realiza o cálculo de caminhos mínimos de todos os nós em ambos os sentidos do tráfego, utilizando o ETX de cada nó de origem.

4.6.2. Otimização de caminhos para comunicação entre os nós da RSSF

Em uma RSSF tradicional com roteamento realizados pelo protocolo RPL, o fluxo de comunicação é otimizado para o tráfego mais comum, que é entre os nós e o 6LBR. O 6LBR, por sua vez, realiza o roteamento para fora da RSSF-6LoWPAN, onde normalmente está localizado o coletor de dados. A comunicação entre os nós é possível em redes com roteamento RPL, porém os dados devem trafegar até um nó ancestral comum que possua a informação de roteamento para o nó de destino, sendo o 6LBR em boa parte dos casos para RSSFs que operam com o protocolo RPL no modo *non-storing*.

No caso da rede SD6WSN, quando uma aplicação de sensoriamento necessitar enviar dados para outro nó dentro da mesma RSSF, não haverá correspondência para este endereço de destino na tabela de fluxos. Como não há uma entrada na tabela de fluxos que determine o encaminhamento desse pacote, o agente SD6WSN envia uma mensagem *Packet-in* para o controlador, como descrito na Seção 4.4.4. Quando o controlador recebe uma mensagem *Packet-in*, ele dispara o cálculo do caminho de menor custo entre os nós de origem e destino, tanto de ida como de volta, para prover a bidirecionalidade da comunicação. Após este cálculo,

todos os nós pertencentes ao caminho de custo mínimo entre a origem e o destino receberão novas entradas em suas respectivas tabelas de fluxo.

A Figura 4.17 exemplifica um caminho mínimo entre dois nós da mesma RSSF (no caso, os nós 3 e 5), formado pelas setas com linhas sólidas.

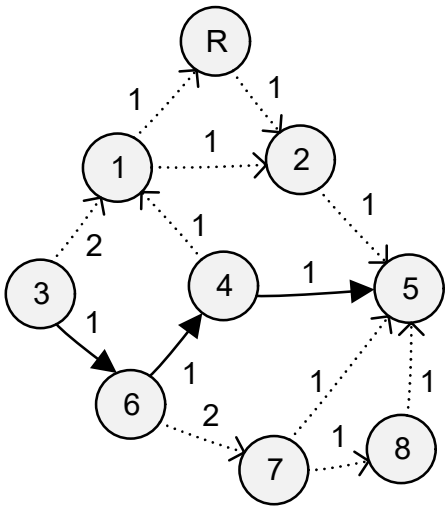

Figura 4.17 – Exemplo de um caminho mínimo entre dois nós da mesma RSSF

4.6.3. Distribuição de tráfego por caminhos alternativos

O protocolo RPL constrói um conjunto de pais alternativos para cada enlace principal, de acordo com a qualidade dos enlaces de seus vizinhos, porém o tráfego é sempre direcionado para o pai preferencial. No caso da arquitetura SD6WSN, onde o controlador centralizado possui a visão completa da rede e determina o caminho para cada fluxo, se existirem enlaces alternativos com qualidade suficiente, um algoritmo baseado no tráfego médio de cada fluxo pode prover o balanceamento de carga por caminhos menos congestionados, proporcionando assim um aumento da capacidade total de transmissão de dados da rede. No exemplo mostrado na Figura 4.18, o nó 3 pode dividir o tráfego destinado ao 6LBR (nó R) por dois caminhos, ou pelo nó 1 (com setas sólidas) ou pelo nó 4 (com setas pontilhadas).

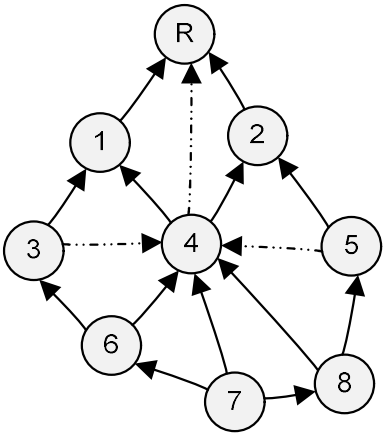

Figura 4.18 – Exemplo de balanceamento de tráfego por dois nós.

As RSSFs que possuem o comportamento padrão, com uma concentração de tráfego dirigido ao 6LBR podem se beneficiar desta aplicação ao se evitar que o tráfego vindo dos nós mais distantes se concentre em apenas um nó antes do 6LBR.

4.6.4. Planejamento de potência de transmissão por fluxo

A escolha da potência de transmissão de cada nó pertencente a um caminho para um fluxo tem a função de otimizar a utilização do meio, diminuindo a interferência de uma transmissão nos nós adjacentes. Na especificação das entradas da tabela de fluxos na arquitetura SD6WSN foi prevista uma entrada para a definição do nível de potência no qual os pacotes de cada fluxo serão transmitidos. PENNA *et al*. [PEN2017] apresentam em seu artigo um algoritmo para a determinação da potência de transmissão de cada nó em redes AMI, que pode ser utilizado no desenvolvimento dessa aplicação.

4.6.5. Separação lógica de tráfego

Em alguns casos práticos, há o interesse de se isolar logicamente determinados pacotes oriundos de mesmos nós de origem, de tal forma que eles não trafeguem por alguns nós intermediários, analogamente às VLANs (*Virtual Local Area Networks*) presentes em redes cabeadas formadas por *switches*. Os objetivos dessa separação lógica são basicamente de evitar que pacotes com informações sensíveis trafeguem por nós intermediários localizados em locais inseguros e para forçar o tráfego de pacotes prioritários ou com alta exigência de disponibilidade somente por nós alimentados por energia ininterrupta.

As características de cada nó podem ser cadastradas no banco de dados da aplicação e a elas atribuídas pesos que influenciarão a tomada de decisão de cálculo de menores caminhos

para tráfegos com características diferentes do padrão, que tem os caminhos calculados e entradas instaladas nas tabelas de fluxo por outras aplicações.

4.7. Conclusão

Esse capítulo apresentou a proposta do *framework* SD6WSN, com a descrições dos componentes das arquiteturas física e lógica, a comunicação entre os componentes e os processos envolvidos para a descoberta e manutenção de topologia lógica da SD6WSN. Foram descritas as mensagens do protocolo SD6WSNP apresentado neste trabalho e a sequência das mesmas para que os processos básicos de formação da rede possam ser efetuados, bem como a sua utilização em aplicações de otimização do tráfego de dados na RSSF. Das aplicações que foram idealizadas para a utilização desta arquitetura, a de cálculo de caminhos mínimos foi mais detalhada por ser a utilizada nos experimentos de validação descritos no capítulo 6 e por mostrar a flexibilidade que existe na implantação de novas aplicações, com a utilização de algoritmos que possam trazer melhor desempenho e novas funcionalidades às RSSFs. O próximo capítulo descreve com maiores detalhes as arquiteturas de *software* e *hardware* dos nós empregados na prova dos conceitos introduzidos neste capítulo e a sua implantação em ambientes simulados e reais.

Capítulo 5

Implementação do *Framework* na Plataforma Contiki

Este capítulo detalha como os componentes do *framework* SD6WSN foram implementados em uma plataforma de desenvolvimento e a montagem do ambiente para os experimentos de validação dos conceitos apresentados. A plataforma de desenvolvimento escolhida foi a do sistema operacional de código aberto Contiki[2], que possui todas as bibliotecas em linguagem "C" necessárias para a escrita do agente SD6WSN e também inclui o emulador COOJA [OST2006]. Esse emulador possibilita a montagem de uma RSSF totalmente emulada, com *motes* e roteadores de borda que executam *firmwares* compilados para a arquitetura de *hardware* encontradas em *motes* disponíveis comercialmente. Para o desenvolvimento do componente controlador e da aplicação "cálculo de caminhos mínimos", que são executados em um servidor Linux, a linguagem Node.js[3] foi escolhida por possuir bibliotecas disponíveis em forma de código aberto que implementam clientes e servidores CoAP e o algoritmo Dijkstra.

Todos os códigos de programação e as instruções para a instalação do ambiente de desenvolvimento e das bibliotecas utilizadas neste projeto estão disponíveis no sítio Github.com[4].

[2] http://contiki-os.org/ – Acesso em 10 de janeiro de 2018.
[3] https://nodejs.org/ – Acesso em 10 de janeiro de 2018.
[4] https://github.com/marciolm/sd6wsn – Acesso em 10 de janeiro de 2018.

5.1. Arquitetura de *software*

Os nós SD6WSN implementam as seguintes funções, como mostrado na Figura 4.1: agente SD6WSN, roteamento do plano de controle, encaminhamento no plano de dados, roteador de borda e aplicação de sensoriamento. A aplicação de sensoriamento se refere à função do *mote* não relacionada à comunicação de dados e, portanto, não será abordada nesse capítulo. Para fins de avaliação de desempenho da rede, uma aplicação de geração de pacotes UDP foi desenvolvida, simulando o envio de pacotes de dados de uma aplicação de sensoriamento real.

5.1.1. Agente SD6WSN

O Contiki [DUN2004] é um sistema operacional especialmente desenvolvido para sistemas embarcados conectados em rede e possui em seu código suporte completo às redes 6LoWPAN e ao protocolo RPL entre outras funcionalidades para sua utilização em RSSFs com dispositivos com pouca capacidade de processamento e memória. Em cima desta base podem ser desenvolvidas aplicações específicas em linguagem C, como é o caso do agente SD6WSN apresentado neste trabalho.

A integração entre o agente SD6WSN e o sistema operacional se dá através das APIs existentes no código fonte do sistema operacional. O código está organizado em uma árvore de diretórios conforme mostrado na Fig. 5.1.

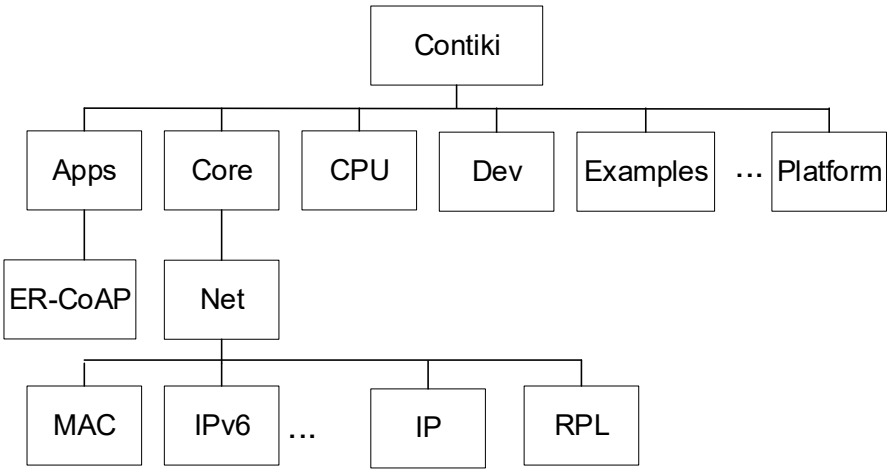

Figura 5.1 – Árvore de diretórios do S.O. Contiki

O ramo de diretório denominado "/core/net/" contém as funções de tratamento da camada de enlace, inclusive as relativas às métricas de qualidade de enlace, como as funções de cálculo de ETX e medidas de RSSI. No ramo de diretório "/core/net/rpl" estão os códigos que implementam a lógica de funcionamento do protocolo RPL e em "/core/net/ipv6", está o código referente ao IPv6 e ao 6LoWPAN, e outras funções, como a de descoberta de vizinhança.

O ER-CoAP [KOV2011] é a implementação padrão para o Contiki do servidor CoAP e já faz parte de sua árvore padrão, dentro do diretório "Apps". Ele pode ser compilado conjuntamente com o sistema operacional para a geração do *firmware* que é gravado nos *motes* suportados por diversas plataformas de *hardware*, que têm as suas definições de arquitetura nos diretórios "Platform" e "CPU". Os *drivers* dos dispositivos, como os dos rádios e dos adaptadores Ethernet, ficam armazenados no diretório "Dev".

O agente SD6WSN é implementado pelo programa principal "Agente-SD6WSN.c", e se integra com o servidor ER-CoAP por meio de chamadas de função. O agente implementa, através de módulos específicos, as funcionalidades do protocolo SD6WSNP, denominadas *resources*, as quais realizam a interface com as funções do sistema operacional. As mensagens do protocolo SD6WSNP são endereçadas aos módulos específicos de acordo com suas URIs. A Figura 5.2 ilustra a integração entre o agente SD6WSN, a aplicação ER-CoAP e os *resources* para as funcionalidades SD6WSN e a interface com as funções do sistema operacional.

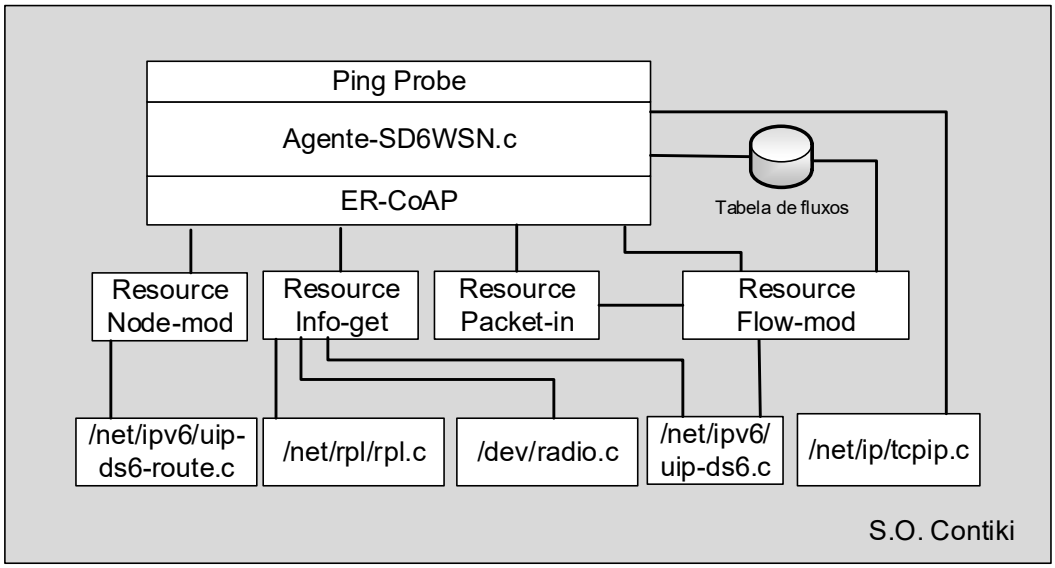

Figura 5.2 – Integração do agente SD6WSN com o S.O. Contiki

O mapeamento dos URIs definidas no protocolo SD6WSNP com as funções que executam as tarefas de cada mensagem do protocolo é realizado no módulo ER-CoAP compilado juntamente com o agente SD6WSN. O código da Tabela 5.1 mostra a forma como os *resources* são mapeados para as URIs definidas no protocolo:

Tabela 5.1 – Mapeamento dos recursos para URIs do protocolo SD6WSNP

```
1 rest_activate_resource(&res_etx,"sd6wsn/info-get/nbr-etx");
2 rest_activate_resource(&res_txpower,"sd6wsn/info-get/txpower");
3 rest_activate_resource(&res_rssi,"sd6wsn/info-get/rssi");
4 rest_activate_resource(&res_battery,"sd6wsn/info-get/battery");
5 rest_activate_resource(&res_location,"sd6wsn/info-get/location");
6 rest_activate_resource(&res_packetin,"sd6wsn/packet-in");
7 rest_activate_resource(&res_flow_mod,"sd6wsn/flow-mod");
8 rest_activate_resource(&res_node-mod,"sd6wsn/node-mod");
```

Os *resources* que implementam as mensagens do SD6WSNP são:

- *Resource Info-get*: provê a leitura dos valores de ETX dos vizinhos de um nó. Utiliza a opção CoAP *observe*, com o envio de notificações ao controlador quando ocorrerem mudanças significativas nos valores medidos para o ETX, ou quando os vizinhos do nó se alteram (*nbr-etx*). Também permite a leitura da potência de transmissão para cada fluxo definido na tabela de fluxos (*txpower*) e a leitura de outros parâmetros como o RSSI, a tensão da bateria (*battery*) e a localização geográfica do nó (*location*);
- *Resource Packet-in*: informa ao controlador a entrada no nó de um pacote sem correspondência na tabela de fluxos. Esse *resource* possui a opção *observe* habilitada, a qual será acionada sempre que dados para um fluxo não existente são identificados;
- *Resource Flow-mod*: recebe mensagens do controlador para inserir ou apagar entradas na tabela de fluxos. Um *parser* decodifica a mensagem REST do protocolo SD6WSNP recebida do controlador e, caso a operação especificada seja "*insert*", ele insere os demais campos da mensagem em entradas da tabela de fluxo. Caso a operação a ser executada seja "*delete*", a entrada da tabela com o "*flowid*" informado é apagada;
- *Resource Node*-mod: envia notificações (*observe*) ao controlador quando um novo nó é inserido ou removido da RSSF.

5.1.2. Roteamento no plano de controle

O roteamento no plano de controle utiliza o protocolo RPL, que determina as rotas de encaminhamento dos pacotes do protocolo SD6WSNP dentro da RSSF. A partir do 6LBR, o roteamento dos pacotes IPv6 até o controlador e o restante da rede é realizado pelos protocolos adotados pela rede IPv6, como o OSPFv3, BGP e etc.

As funções de roteamento RPL nos nós da RSSF são realizadas pelas bibliotecas RPL nativas do Contiki, localizadas em /core/net/rpl, e pela aplicação "border-router.c" dentro do 6LBR, que também utiliza as bibliotecas do Contiki para implementar as funções de raiz do DODAG RPL.

5.1.3. Encaminhamento no plano de dados

O encaminhamento no plano de dados é responsável por uma série de tarefas:

- Inspeção do tráfego de entrada do nó;
- Decisão de encaminhamento dos pacotes se o pacote é do plano de dados ou controle (os pacotes do plano de controle são passados para função de roteamento do plano de controle);
- Encaminhamento de pacotes do plano de acordo com a tabela de fluxos, incluindo a alteração de potência de transmissão do nó, caso esteja determinado na entrada correspondente;
- Encaminhamento ao controlador dos cabeçalhos dos pacotes do plano de dados via mensagens "*Packet-in*".

A lógica principal de encaminhamento de pacotes no Contiki está no ramo "/core/net/ip" da árvore de diretórios, principalmente no programa "tcpip.c". Neste programa foi inserido o trecho de código mostrado na Figura 5.2, que tem o objetivo de identificar os campos do cabeçalho do pacote que está no *buffer* de saída do nó e chamar a função presente no agente SD6WSN denominada "*get_next_hop_by_flow*", utilizando os dados dos campos do cabeçalho como seus argumentos. Caso o pacote seja identificado como UDP (linha 3), as portas de origem e destino são identificadas (linhas 4 e 5) e a função é chamada (linha 6), com uma variável de retorno "nexthop", que contém o endereço IPv6 link-local do nó do próximo salto.

Tabela 5.2 – Código para a chamada de função de determinação de endereço de próximo salto no programa "tcpip.c" do Contiki

```
#if SDWSN
  uint8_t proto_out = *((uint8_t *)UIP_IP_BUF + 40);
  if(proto_out==17) {
    uint16_t udpsrcport = (*((uint8_t *)UIP_UDP_BUF + 8) << 8) |
    *((uint8_t *)UIP_UDP_BUF + 9);
    uint16_t udpdstport = (*((uint8_t *)UIP_UDP_BUF + 10) << 8) |
    *((uint8_t *)UIP_UDP_BUF + 11);
    nexthop=get_next_hop_by_flow(&UIP_IP_BUF->srcipaddr,
    &UIP_IP_BUF->destipaddr,udpsrcport,udpdstport,proto_out);
  }
#endif
```

O endereço IPv6 de retorno da função "*get_next_hop_by_flow*" depende da ação presente na entrada da tabela de fluxos, que pode ser:

- Um endereço NULL, que força o roteamento RPL a calcular o endereço de *next-hop*. É retornado quando o pacote se tratar de uma mensagem do plano de controle ou resultante de uma correspondência que tenha ação *CPForward*;
- Um endereço de *loopback*, que causa o descarte do pacote, necessário quando a correspondência do pacote for com uma entrada que possua a ação *drop*;
- Ou o endereço IPv6 link-local de próximo salto definido pelo parâmetro *nhipaddr* da entrada da tabela de fluxos, quando houver uma correspondência que possua uma ação *Forward* definida.

O código que implementa a função "*get_next_hop_by_flow*" dentro do agente SD6WSN é apresentado na Tabela 5.3.

Tabela 5.3 – Função "*get_next_hop_by_flow*"

```
1  flow_s flow_table[32];  // Criação da tabela de fluxos
2  uip_ipaddr_t * get_next_hop_by_flow(uip_ipaddr_t
     *srcaddress,uip/_ipaddr_t *dstaddress,uint16_t *srcport,uint16_t
     *dstport,uint8_t *proto){
3    table_pos = 0;
4    if(dstport == 5683 || srcport == 5683 ) { // se pacote do plano de
5      return NULL;                            // controle, retorna NULL
6    }
7    while(table_pos<=table_entry){
8      if(uip_ipaddr_cmp(dstaddress,&flow_table[table_pos].ipv6dst)) {
```

```
9         if(uip_ipaddr_cmp(srcaddress,&flow_table[table_pos].ipv6src)){
10          if(srcport == flow_table[table_pos].srcport ||
              flow_table[table_pos].srcport == NULL ){
11            if(dstport == flow_table[table_pos].dstport ||
                flow_table[table_pos].dstport == NULL){
12              if(proto == flow_table[table_pos].ipproto ||
                   flow_table[table_pos].ipproto == NULL){
13                 PRINTF("flow found !\n");
14                 break;
15              }
16            }
17          }
18        }
19      }
20      table_pos++;
21    }
22    if(table_pos>table_entry) {
23              packet_in_process(); // não encontrado match na tabela
24    }else{
25      if(flow_table[table_pos].action == 0 ) {   //action = Forward
26        return &flow_table[table_pos].nhipaddr; // retorna next-hop
27      } else {
28        if(flow_table[table_pos].action == 2 ) { //action = Drop
29          return loopback_address;
30        } else {                                  //action = CPForward
31          return NULL;                       // remete ao roteamento RPL
31        }
33      }
34    }
35  }
```

A tabela de fluxos é criada por meio de uma variável global (na linha 1) com o tipo definido pela *struct* que tem a composição mostrada na Tabela 5.4, segundo a especificação apresentada na Seção 4.3.1. O tamanho da tabela de fluxos foi fixado inicialmente em 32 entradas devido à limitação de capacidade da RAM (de 16 kbytes) do *mote* "Wismote" empregado nas simulações. Para outros modelos de *motes* das classes 1 ou 2, esta tabela pode ter seu tamanho aumentado, o que será necessário no caso de RSSFs com número elevado de saltos e de nós.

O código desta função, que foi implementado mais simplificadamente para fins de prova de conceito, faz uma busca sequencial na tabela de fluxos para a determinação do endereço de próximo salto do pacote a ser encaminhado no plano de dados. Primeiramente é identificada se a porta UDP de origem é do protocolo CoAP do plano de controle (linhas 4 a 6), o que retorna um valor "nulo", para que o roteamento prossiga pelo RPL. Em seguida, na linha 8, é iniciada

a comparação dos campos do cabeçalho do pacote a ser enviado com a tabela de fluxos do nó. Ela também contempla o tratamento de campos “curinga” (nas continuações das linhas 10, 11 e 12), para a criação de fluxos menos específicos como, por exemplo, qualquer porta de origem ou destino. Uma implementação mais completa usaria uma busca mais eficiente do que a adotada (como por meio de *hash-tables*) e analisaria todos os campos da tabela de fluxos que não foram utilizados na prova de conceito. Estas tarefas estão destinadas a um trabalho futuro.

Tabela 5.4 – Definição da tabela de fluxos

```
1   typedef struct flow_s {
2         uint8_t flowid;
3         uip_ipaddr_t ipv6src;
4         uint8_t srcmask;
5         uip_ipaddr_t ipv6dst;
6         uint8_t dstmask;
7         uint16_t srcport;
8         uint16_t dstport;
9         uint8_t ipproto;
10        uint8_t action;
11        uip_ipaddr_t nhipaddr;
12        uint8_t txpwr;
13  }flow_s;
```

5.1.4. Mecanismo de prova ativa para atualização do ETX

Em algumas implementações do RPL (como na do Contiki), existe um mecanismo opcional de prova ativa da qualidade dos enlaces, denominada “RPL Probing”, que utiliza mensagens *ICMPv6* do tipo DAO ou DIS para promover uma geração de tráfego *unicast* e a consequente atualização dos valores de ETX. Este mecanismo somente envia mensagens para os pais preferenciais do nó, ficando os demais enlaces sem atualização.

Para se evitar a utilização de mensagens RPL na prova de qualidade e prover a atualização do ETX para todos os vizinhos de cada nó, o mecanismo proposto neste módulo (identificado como “*ping probe*” na Figura 5.2) utiliza mensagens *ICMPv6* do tipo *Echo-Request* (que são as mesmas utilizadas pela aplicação "*ping*"). Por padrão, o Contiki retorna para cada *Echo-Request* uma mensagem de *Echo-Reply*, o que ocasiona a atualização do parâmetro de ETX também no nó vizinho para qual o *Echo-Request* foi enviado. A Figura 5.3 exemplifica o processo, com o nó de número 4 enviando mensagens de “*ping*” para todos os nós vizinhos descobertos pelo protocolo RPL. Os demais nós da rede repetem o mesmo processo periodicamente.

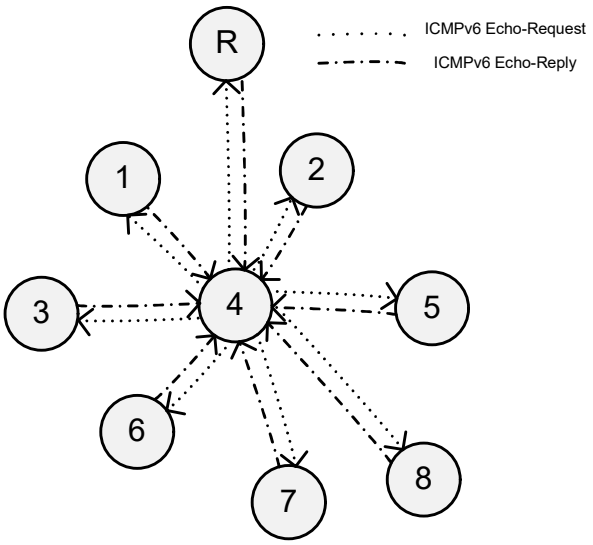

Figura 5.3 – Processo de prova ativa de enlaces com o uso de mensagens "*ping*"

O processo de envio periódico de provas ativas ("*pings*") utiliza o mesmo intervalo de 120 segundos definido pelo "RPL *Probing*" do Contiki, com uma variação randômica de mais ou menos 20 segundos, para prevenir a sincronização entre os nós, o que poderia ocasionar picos de utilização da rede nos momentos de prova.

O envio de provas ativas para cada vizinho de um nó possui um intervalo de 0,5 segundos entre eles, também para evitar picos de utilização de banda.

5.1.5. Módulo de geração de tráfego

Para os fins de avaliação de desempenho da rede, um módulo de geração de pacotes UDP foi incluído na compilação do *firmware* dos *motes* com o objetivo de simular uma "aplicação de sensoriamento". Esse módulo é composto de clientes, instalados nos nós da RSSF, onde pacotes UDP são gerados com destino para um servidor instalado no 6LBR, que os recebe e gera pacotes de retorno. A finalidade desta aplicação é gerar tráfego de dados no plano de encaminhamento, como se fossem dados de sensoriamento, e assim possibilitar a medição de parâmetros de rede, como o PRR e o tempo decorrido entre a transmissão e retorno dos pacotes. Esta aplicação foi utilizada nos experimentos descritos no Capítulo 6.

5.2. Arquitetura de *hardware*

Os nós SD6WSN tipicamente são construídos com circuitos integrados de RF projetados para o uso em RSSFs, aderentes aos padrões IEEE 802.15.4 para a frequência de 2,4 GHz e IEEE 802.15.4g para frequências abaixo de 1 GHz. Alguns fabricantes agrupam no mesmo circuito integrado outros componentes como processador, RAM, memória Flash, conversores A/D, UARTs, portas de entrada e saída digitais e etc., formando o que se denomina SoC. Esse agrupamento tem o intuito de diminuir o custo e consumo de energia, além de simplificar e padronizar o desenvolvimento de aplicações para o sistema. Um sistema composto pelo transceptor RF, processador e periféricos é comumente chamado de *mote*.

A padronização dos projetos de *motes* pela utilização de SOCs, que seguem os projetos básicos fornecidos pelos fabricantes, permitiu o melhor suporte do Contiki a estas arquiteturas padronizadas, definidas na árvore de desenvolvimento do Contiki nos diretórios /Platforms e em /CPU.

Dentre as diversas plataformas de *hardware* suportadas pelo Contiki, algumas foram lançadas mais recentemente e com maiores capacidades de processamento e memória. Estes fatores são importantes para que o agente SD6WSN possa ser executado conjuntamente com a aplicação de sensoriamento nativa ao nó. Entre elas, pode-se citar as baseadas nos SOCs da Texas Instruments CC2538[5] para a operação na banda de 2,4 GHz, o qual possui 512 kbytes de memória *Flash* e 32 kbytes de RAM e o CC1310[6] para operação em bandas abaixo de 1 GHz, com 128 kbytes de memória *Flash* e 20 kbytes de RAM. Entretanto, essas plataformas não puderam ser utilizadas nos experimentos de validação apresentados no capítulo 6, porque ainda não foram incluídas no emulador COOJA.

A função de comunicação denominada “Roteador de Borda” está presente em nós particulares da rede, onde ocorre a conversão do protocolo da rede externa (IP) para o protocolo de transporte específico da RSSF. Esta seção apresenta as quatro principais opções para construção de um 6LBR baseado no Contiki, cada uma com uma divisão diferente das camadas do modelo OSI entre os componentes do roteador.

[5] http://www.ti.com/product/CC2538 – Acesso em 10 de janeiro de 2018.
[6] http://www.ti.com/product/CC1310 – Acesso em 10 de janeiro de 2018.

5.2.1. 6LBR em um sistema Linux conectado a um *mote* IEEE 802.15.4

A primeira opção utiliza um hospedeiro com um sistema operacional Linux que "hospeda" um segundo sistema operacional (no caso o Contiki), encarregado das funções de comunicação para a RSSF. O Contiki implementa uma pilha de protocolos própria, denominada µIPv6, que contempla na camada 3 os protocolos IPv6 e 6LoWPAN, e na camada 4 os protocolos ICMPv6, o TCP e UDP. Junto ao sistema operacional Contiki também são compilados programas da camada de aplicação, como o roteador RPL e servidores HTTP e CoAP, entre outros.

A aplicação Contiki que desempenha a função 6LBR neste modelo é denominada "native-border-router.c"[7], cujo código é compilado para a CPU do computador hospedeiro.

As funções de camada física e de enlace (MAC) ficam a cargo de um *mote* externo, que se liga ao hospedeiro por meio de uma conexão serial. Neste *mote*, normalmente constituído por um SoC onde há um rádio padrão IEEE 802.15.4 e um microcontrolador, também é carregado o Contiki, responsável apenas pelas funções de camada física e MAC, e um programa denominado "slip-radio.c"[8] para estabelecer a conexão serial com o hospedeiro. Os *drivers* das camadas MAC e RDC (*Radio Duty Cycle*) são escolhidos de acordo com a especificação da RSSF. A conexão serial, denominada SLIP (*Serial Line* IP), é descrita pelos desenvolvedores como sendo "IEEE 802.15.4 *Over Serial Line*", pois são transmitidos quadros IEEE 802.15.4 pela conexão serial e não quadros IP.

A comunicação entre a aplicação Contiki "border-router.c" (que utiliza a própria pilha de protocolos do Contiki) com a pilha IPv6 do sistema operacional hospedeiro é realizada através uma interface túnel (Tun0), definida no momento da inicialização do programa "border-router.c". Na linha de comando deste programa é definida qual a interface serial (ttyUSB0, por exemplo) onde está o *mote* com o programa "slip-radio" e qual a interface túnel do Linux que será utilizada.

O computador que hospeda o Contiki com a aplicação de roteamento RPL também possui uma interface Ethernet para a comunicação com as demais redes IPv6 e a Internet. A Figura 5.4 ilustra essa opção de configuração, onde pode-se observar a presença das duas pilhas de protocolos, uma no programa "*border-router*" e outra do sistema operacional hospedeiro, e a comunicação entre as duas pilhas sendo realizada pela interface virtual do tipo túnel.

[7] https://github.com/contiki-os/contiki/tree/master/examples/ipv6/native-border-router – Acesso em 10 de janeiro de 2018.
[8] https://github.com/contiki-os/contiki/tree/master/examples/ipv6/slip-radio – Acesso em 10 de janeiro de 2018.

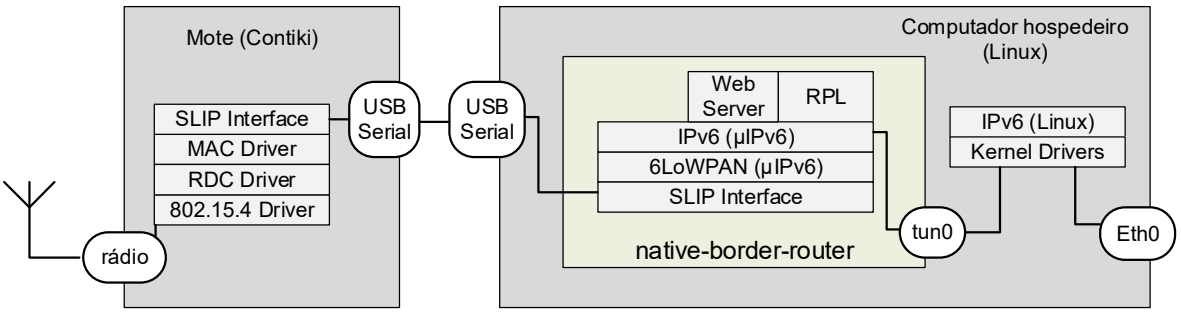

Figura 5.4 – 6LBR na opção Linux conectado a um *mote*

Uma das vantagens deste modelo é a possibilidade de uso de *motes* de menor capacidade de processamento e memória com a função "*slip-radio*", pois somente a camadas física, de enlace e RDC estão presentes. As camadas que necessitam de maior poder processamento são tratadas pelo programa executado no computador hospedeiro, que normalmente possui um processador muito superior aos encontrados nos *motes*.

5.2.2. 6LBR em um *mote* conectado a um hospedeiro Linux

A segunda opção para a construção de um 6LBR é transferir mais funções do computador hospedeiro para dentro do *mote*, além das relativas às da camada física, MAC e RDC. No ambiente de desenvolvimento do Contiki existe outra versão do programa "border-router.c"[9], diferente da utilizada na Seção 5.3.1, onde as funções das camadas 3 e 4 (6LoWPAN, IPv6 e RPL), além de um servidor HTTP simples, são compilados conjuntamente às funções de camada 1 e 2 existentes no programa "slip-radio.c".

A Figura 5.5 mostra essa alternativa, com o deslocamento das funções para o *mote*, ficando no computador hospedeiro somente um programa denominado "tunslip6.c" responsável pelo estabelecimento de uma conexão SLIP com o *mote* por meio de uma interface serial. Diferentemente da conexão SLIP efetuada pela opção anterior, nesta conexão SLIP trafegam pacotes IPv6, e conecta uma interface túnel Linux (Tun0) diretamente a um processo SLIP do *mote*.

[9] https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-border-router – Acesso em 10 de janeiro de 2018.

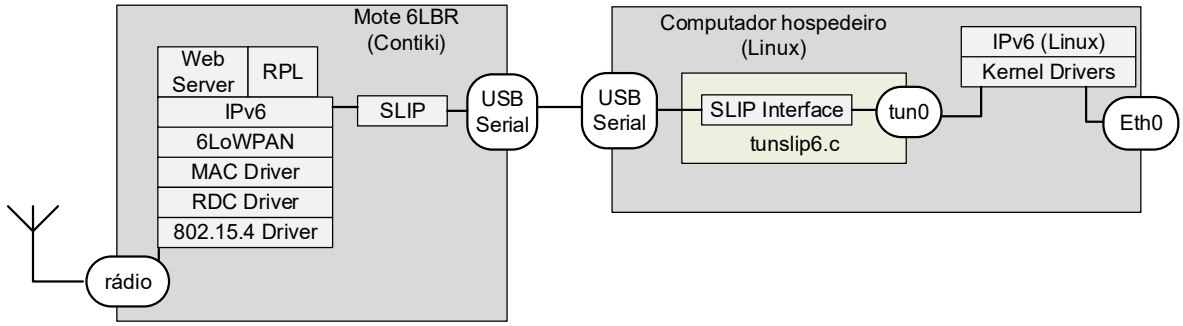

Figura 5.5 – 6LBR na opção *mote* conectado ao computador Linux

A agregação de mais funções exige que os "*motes*" empregados possuam um *hardware* com um melhor processador e maiores quantidades de memória RAM e *Flash*, o que inabilita o uso de *motes* de menor custo. Mesmo para SOCs mais recentes, como o CC2538 ainda existem limitações de processamento decorrentes da exigência de baixo consumo de energia para esses dispositivos, que foram projetados principalmente para serem alimentados por baterias. O computador hospedeiro pode ser simples, como por exemplo, roteadores Wi-fi que executam distribuições Linux OpenWRT[10].

5.2.3. 6LBR em *mote* com interface Ethernet

A terceira opção de configuração é adequada para quando há a disponibilidade de *motes* com maior capacidade de memória e CPU, como os baseados no SoC CC2538, ao qual é acoplado um adaptador Ethernet com interface SPI (*Serial Peripheral Interface*). Isto permite que o *mote* com o Contiki funcione sem um computador hospedeiro. Como a interface SPI é diretamente conectada ao *hardware* do *mote*, não há a necessidade de uma conexão SLIP nesta arquitetura, como mostrado na Figura 5.6.

[10] https://openwrt.org/ – Acesso em 10 de janeiro de 2018.

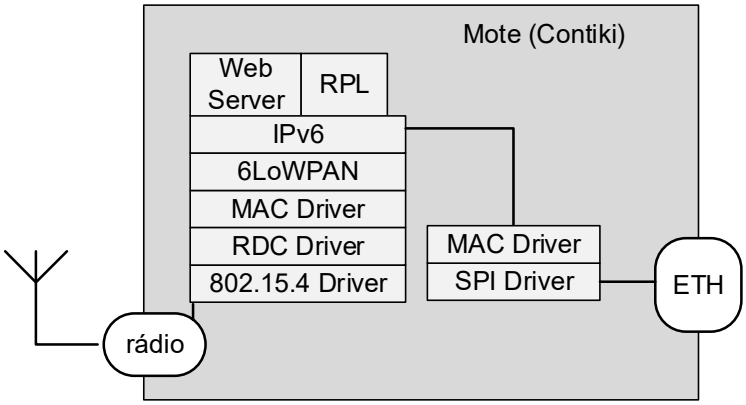

Figura 5.6 – 6LBR na opção *mote* com interface Ethernet

Esta arquitetura está se tornando mais comum com o lançamento de novos SOCs com maior capacidade de memória, como o SoC CC2538 integrado com um módulo Ethernet que utiliza o controlador ENC28j60 da Microchip.

Apesar das vantagens de se ter uma arquitetura integrada, as desvantagens deste modelo residem no pouco espaço de memória para a adição de novos componentes, tal como um servidor HTTP, para a gerência do 6LBR via Web. O desempenho também é afetado pelo pouco poder de processamento em função das limitações inerentes a esta classe de dispositivos de baixo consumo de energia, como a baixa frequência de *clock*, de apenas 32 MHz para o SoC CC2538.

5.2.4. Suporte IEEE 802.15.4 e 6LoWPAN nativo no *kernel* Linux

Alguns projetos, como o "IEEE 802.15.4 *Stack for Linux*"[11] tem como objetivo a implementação da pilha IEEE 802.15.4 e do 6LoWPAN para Linux, porém no presente momento ainda se encontra em estágio de desenvolvimento "alpha".

Este modelo de implementação retira a necessidade de duas pilhas de protocolos, deixando a cargo do Linux todo o processamento das camadas MAC, RDC, 6LoWPAN, IPV6 e RPL, e para um *hardware* externo apenas a camada física, com o rádio se comunicando com o computador Linux por interface SPI. Por estar em estágios iniciais de desenvolvimento, esta alternativa não foi contemplada no presente trabalho.

[11] https://sourceforge.net/projects/linux-zigbee/ – Acesso em 10 de janeiro de 2018.

5.3. Ambiente de desenvolvimento

O Contiki possui em sua estrutura de desenvolvimento[12] diversas bibliotecas, exemplos de aplicações e ferramentas desenvolvidas pela comunidade de *software* livre que podem ser utilizadas para a implementação dos aplicativos, além de um total acesso ao código fonte dos componentes do sistema operacional, como os *drivers* de baixo nível das camadas físicas e de enlace e ao código da implementação do 6LoWPAN e do RPL.

5.3.1. Simulador COOJA

Uma rede de sensores que utiliza o Contiki pode ser simulada com a utilização do simulador COOJA [OST2006]. Uma das características mais importantes do COOJA é de permitir simulações nos níveis de rede, de sistema operacional e no nível de instruções de código de máquina [OST2006]. O COOJA permite a montagem da rede (Figura 5.7) e determinação de parâmetros utilizando *motes* emulados, ou seja, *motes* que emulam o funcionamento do *hardware* real, inclusive rodando o *firmware* exclusivamente compilado para eles. O sítio do Contiki disponibiliza a imagem de uma máquina virtual Linux com todos os *softwares* necessários para a compilação dos *firmwares* dos *motes* e execução da simulação da rede com o COOJA.

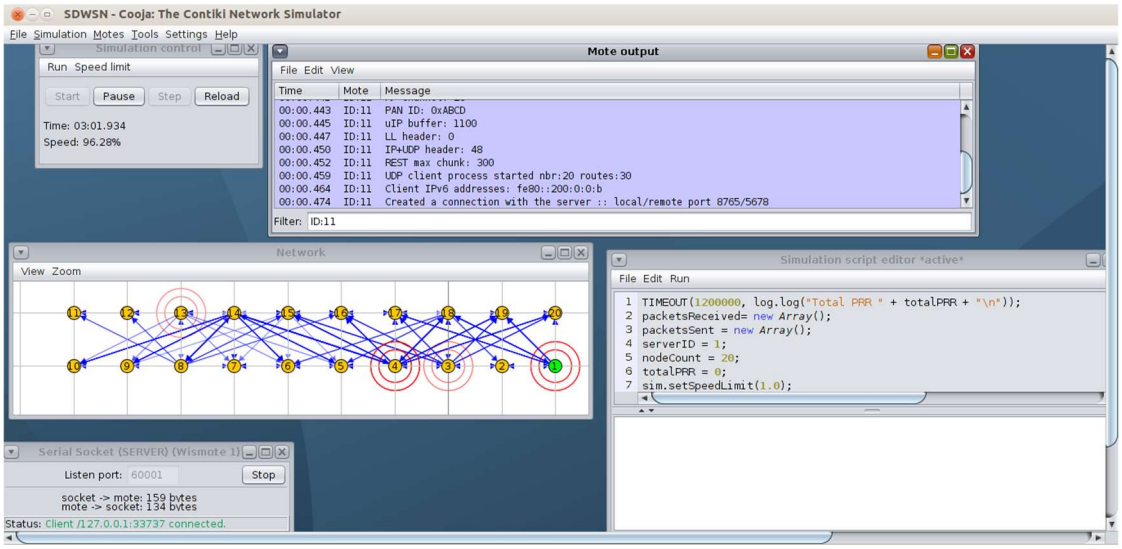

Figura 5.7 – Tela do simulador COOJA

[12] http://contiki-os.org/start.html - Acesso em 10 de janeiro de 2018.

Os pacotes de dados trocados entre os *motes* durante a simulação podem ser capturados e analisados com o analisador de protocolos Wireshark,[13] da mesma forma que em uma rede física.

5.3.2. Ambiente de testes real

A construção do ambiente de testes real baseou-se em *motes* que empregam o SoC do fabricante Texas Instruments (T.I.) da linha CC25xx. O ambiente apresenta como 6LBR um minicomputador *Raspberry Pi 3 model B*[14] equipado com um *mote* USB que possui o chip CC2531 e com o *firmware* Contiki, compilado com o *firmware* SD6WSN *Border Router*, na arquitetura descrita na Seção 5.2.2. A Figura 5.8 mostra um conjunto montado pelo autor, contendo à direita da foto o minicomputador com dois *motes*, um sendo o USB *Border Router*, e um segundo *mote* idêntico, porém carregado com o *firmware* "*sniffer*" CC2531EMK[15], utilizado para a captura de pacotes IEEE 802.15.4 por radiofrequência, utilizado para a análise em tempo real dos pacotes transmitidos pelos dispositivos próximos.

Figura 5.8 – Conjunto para testes em ambiente real da RSSF SD6WSN

[13] https://www.wireshark.org/ - Acesso em 10 de janeiro de 2018.
[14] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ - Acesso em 10 de janeiro de 2018.
[15] http://www.ti.com/tool/cc2531emk - Acesso em 10 de janeiro de 2018.

Para a construção deste *testbed* foram escolhidos *motes* que empregam o SoC modelo CC2538 por possuírem quantidades de memória RAM e memória Flash adequadas para a instalação do S.O. Contiki 3.0 e do agente SD6WSN.

A diferença básica entre a utilização de um *testbed* físico em relação à simulação realizada com o COOJA é a percepção da influência das interferências existentes no meio, tais como redes Wi-Fi operando em frequências sobrepostas às utilizadas pela RSSF, bem como a diminuição do alcance dos rádios quando da existência de obstáculos.

5.3.3. Conclusão

Neste capítulo, em sua Seção 5.1, foram apresentados os módulos de *software* utilizados para a implantação do *framework* SD6WSN no ambiente Contiki, possibilitando a execução dos ensaios no simulador COOJA. Na Seção 5.2 foram descritos os componentes de *hardware*, que tanto podem ser emulados no simulador COOJA quanto construídos com módulos existentes no mercado. Na Seção 5.3, foi descrito o ambiente de simulação utilizado nos experimentos apresentados no capítulo 6 e um exemplo de um possível *testbed* que emprega componentes de *hardware* reais.

A construção física do *testbed* descrito na Seção 5.3.2, formado por um 6LBR e seis nós, teve o objetivo provar a factibilidade da implementação do *framework* apresentado em um ambiente real e utilizando *motes* com arquitetura de *hardware* diferente da utilizada na simulação. No presente trabalho, os ensaios foram exclusivamente realizados em ambiente simulado devido às seguintes restrições encontradas para a execução dos experimentos em ambiente real:

- O alcance de transmissão de cada *mote* é variável e não determinístico, pois depende das obstruções existentes e demais variáveis do ambiente;
- A área necessária para a montagem do *testbed* pode ser muito extensa para alguns cenários;
- A taxa de sucesso de transmissão não é determinística, também depende das condições do ambiente;
- O alcance do *sniffer* é limitado aos nós próximos e, no caso de múltiplos *snifferes*, eles devem ter as referências de tempo sincronizadas;

- Não há acesso aos dados de *log* gerados nas interfaces seriais dos *motes*, como os utilizados para a contabilização dos sucessos de transmissão;
- Necessidade de acesso físico aos *motes* no caso de atualização de *firmware*;
- Os *motes* disponíveis se mostraram insuficientes para a criação de uma topologia que proporcionasse um número de saltos adequado para a avaliação de latência e taxa de sucesso de recepção de pacotes.

Outra vantagem encontrada na utilização do COOJA foi a possibilidade de ajuste de parâmetros do Contiki de uma forma bem mais simples do que em redes reais, onde seria necessária a regravação dos *firmwares* nas memórias *Flash* dos dispositivos compilados à cada mudança efetuada.

Como um trabalho futuro, estuda-se a instalação de um *testbed* fixo para o desenvolvimento de projetos de RSSFs, que contemplem os recursos que foram considerados importantes para a execução dos ensaios, como a atualização remota de *firmware* e acesso às *UARTs* dos dispositivos, além da instalação de *sniffers* que abranjam toda a rede.

Capítulo 6

Avaliação da Arquitetura SD6WSN

Neste capítulo são apresentados os ensaios de validação do *framework* SD6WSN. Adotou-se o ambiente Contiki/COOJA para o desenvolvimento das aplicações embarcadas, o que permite o teste de cada elemento da arquitetura de forma ágil por conter um simulador que executa o *firmware* do dispositivo real. A utilização do simulador COOJA permitiu o *debug* dos *firmwares* instalados nos nós e coleta de resultados de uma forma prática, seja pela saída de UART dos *motes* emulados como pela captura dos pacotes trafegados entre eles nos mesmos formatos que seriam feitos em redes reais.

Os ensaios de validação foram divididos em duas partes, o primeiro apresentando um cenário típico de rede AMI em uma topologia empregadas em redes AMI em áreas residenciais, com nós distribuídos uniformemente ao longo de uma via nos dois lados e com comunicação somente entre os nós e o 6LBR. O segundo ensaio foi realizado com a utilização de uma topologia do tipo "grade", para a formação de uma rede do tipo *peer-to-*peer, onde o tráfego gerado pelo nós eram direcionados para outros nós da mesma RSSF.

No primeiro ensaio comparou-se o desempenho do *framework* SD6WSN com o RPL nos quesitos de latência média de *round-trip* e PRR entre cada nó e o 6LBR, além do número de mensagens de controle utilizadas pelo RPL e pelo SD6WSNP. No segundo ensaio foi realizada uma comparação da latência média da comunicação (em um sentido) entre nós escolhidos aleatoriamente dentro da RSSF para os caminhos definidos pelo RPL e pela aplicação de "otimização de caminhos dentro da RSSF", descrita na Seção 4.6.2.

Os *motes* simulados foram do modelo "Wismote[16]" com 16 kbytes de RAM e 128 kbytes de memória flash, de arquitetura MSP430[17] e transceptor CC2520[18] de 2,4 GHz. Eles foram escolhidos por possuírem RAM um pouco maior aos demais disponíveis no COOJA.

As simulações foram realizadas em uma máquina virtual Linux Ubuntu 14.4 em um computador com processador Intel Core I7-6500U com 8 Gbytes de RAM. O ambiente contava com a plataforma Contiki completa que inclui o compilador para a CPU MSP430 e o simulador COOJA. Na mesma máquina hospedeira Linux onde foi executado o COOJA também foram instalados os pacotes da linguagem Node.js e as bibliotecas CoAP e Dijkstra, necessárias para a execução do controlador SD6WSN e das aplicações que realizaram os cálculos dos caminhos dentro da RSSF.

6.1. Ensaios no cenário AMI

Os ensaios deste cenário tiveram o objetivo de demonstrar o funcionamento do framework SD6WSN com a aplicação de "cálculo de caminhos mínimos", que provê a conectividade mais comum em RSSFs, dos nós ao 6LBR e vice-versa. Em RSSFs 6LoWPAN que não adotam o paradigma SDN, esse roteamento de pacotes é provido pelo RPL, com o qual os resultados de desempenho resultantes da utilização da aplicação SD6WSN foram comparados.

6.1.1. Metodologia

O posicionamento dos nós foi escolhido de tal forma a criar um cenário de rede NAN encontrado em redes AMI e *Smart Grids* [KUZ2014], que empregam redes 6LoWPAN para a comunicação entre os medidores de consumo de energia e os MDC (*Meter Data Colector*), que realizam a coleta dos dados enviados pelos medidores. A Figura 6.1 ilustra um cenário AMI típico de medição de energia, onde existe um 6LBR para cada conjunto de nós, que tem a conexão até um centro de medição efetuada por meio de enlaces de fibra óptica. Neste centro, onde ficam localizados os servidores MDC, também está localizado o servidor que hospeda as instâncias dos controladores e das aplicações SD6WSN, sendo uma instância para cada RSSF-SD6WSN existente.

[16] http://www.wismote.com/products.html - Acesso em 10 de janeiro de 2018.
[17] http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview/overview.html - Acesso em 10 de janeiro de 2018.
[18] http://www.ti.com/product/CC2520 - Acesso em 10 de janeiro de 2018.

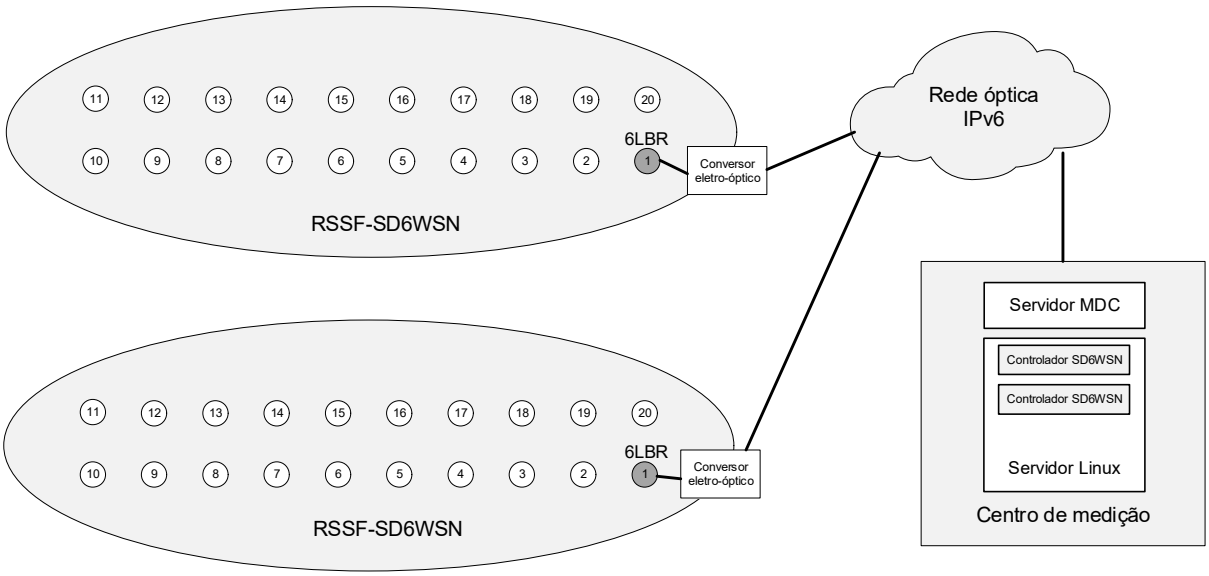

Figura 6.1 – Cenário típico AMI

A avaliação dos resultados obtidos foi feita comparando-se o desempenho da arquitetura proposta com RSSFs convencionais formadas por *motes* também Contiki, porém exclusivamente com roteamento RPL. Os parâmetros de comparação foram o PRR e a latência entre os nós.

Para fins de avaliação do impacto que as mensagens do SD6WSNP causam no tráfego total da RSSF, foi inclusa uma comparação entre o número de mensagens deste com o número de mensagens trocadas pelo RPL.

Para compor o cenário típico de redes AMI em bairros residenciais, ou seja, em uma rua com medidores de energia em ambos os lados, foi utilizada uma topologia do tipo grade com um nó fazendo o papel de 6LBR, similar a um dos ramos da Figura 6.1.

6.1.2. Ambiente de simulação

A grade utilizada na simulação foi formada por duas linhas paralelas com distância entre os nós de 10 m, contendo 20 nós ao todo, conforme mostrado na Figura 6.2. Nesta topologia, o nó “1” representa o 6LBR instalado em um dos vértices e os demais são os nós de sensoriamento.

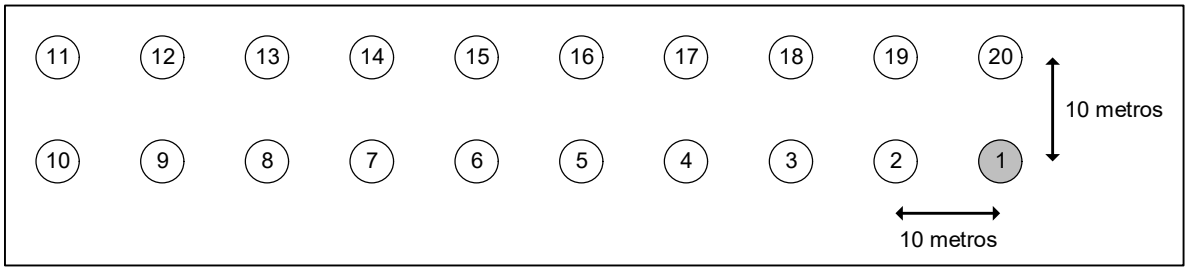

Figura 6.2 – Topologia utilizada nas simulações

Uma vantagem da utilização deste cenário é a possibilidade de alteração da profundidade da rede, com o aumento do número de saltos através da mudança no alcance de cada um dos nós. Em cenários reais a variação de alcance é devida à diversos fatores, entre eles a atenuação e as reflexões do sinal transmitido. Neste estudo foi escolhida a realização dos experimentos tendo como parâmetro a variação do alcance dentro da mesma topologia pela possibilidade de generalização do cenário, pois apresenta o mesmo efeito na simulação que teria a variação da distância entre os nós no cômputo de número de saltos entre os nós e o 6LBR.

A execução das simulações em cima da topologia escolhida se dividiu em quatro cenários, onde se variou o alcance de transmissão de todos os *motes* na configuração do modelo UDGM (*Unit Disk Graph Medium*), disponível para o COOJA: 25m, 50m, 100m e 150m, com alcance de interferência de duas vezes o alcance de transmissão escolhido. Os demais parâmetros do modelo UDGM utilizados foram: taxa de sucesso de transmissão de 75 % e a taxa de sucesso de recepção de 100 %. Foi utilizada uma taxa de sucesso de transmissão menor que 100 % para forçar a perda de pacotes na rede e assim provocar alterações nos valores de ETX durante a simulação.

A comunicação entre o COOJA e a máquina hospedeira se deu através de uma conexão SLIP provida pelo programa “tunslip6.c” apresentado na Seção 5.2.2, que provê a comunicação entre a interface “tun0” do computador hospedeiro e o 6LBR simulado, através de um soquete serial provido pelo COOJA, que por sua vez simula a interface serial USB presente na Figura 5.5. A Figura 6.3 ilustra a integração da simulação COOJA com o controlador e a aplicação de cálculo caminhos mínimos executados no computador hospedeiro.

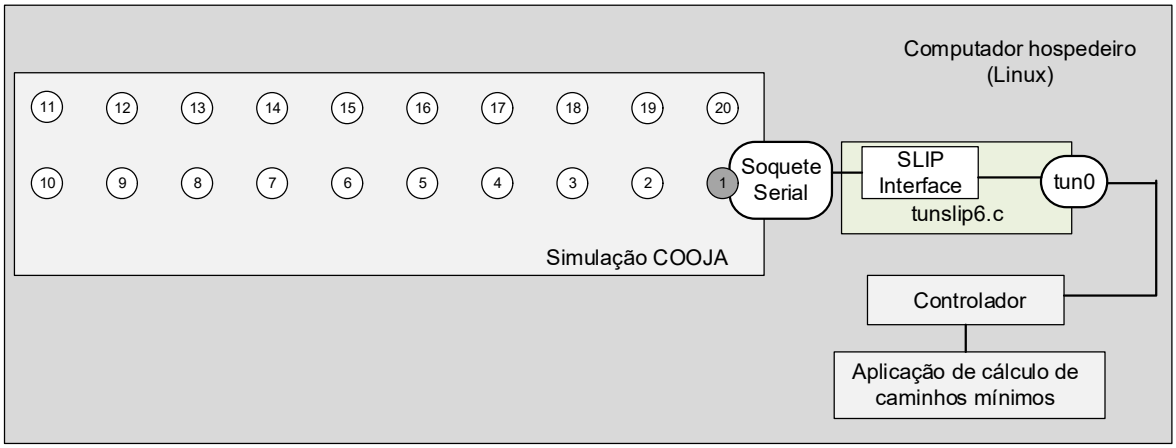

Figura 6.3 – Integração entre a simulação COOJA e o computador hospedeiro

Para as coletas de dados de PRR e latência, foram desenvolvidos *firmwares* que empregam o *framework* SD6WSN para *motes* e para o 6LBR contendo também a aplicação de geração de tráfego (descrita na Seção 5.1.5), que simula a geração de tráfego de uma aplicação real de sensoriamento. Os códigos-fonte dos componentes das simulações, inclusive o código do controlador e a aplicação de cálculo de caminhos mínimos, além dos *scripts* COOJA contendo os cenários, estão disponíveis em uma área criada para este projeto no sítio Github[19].

Nos nós foram instalados os geradores de tráfego e no 6LBR foi instalada a aplicação servidora, que recebe os pacotes UDP e os retorna para os nós de origem. As temporizações da aplicação de geração de tráfego foram ajustadas para que a geração de tráfego se inicie 180 segundos após a inicialização do *mote*, permitindo que os processos RPL e SD6WSN já estejam estabilizados após a entrada no novo nó.

Foi estipulado o envio de um pacote UDP com *payload* de 20 bytes a cada 30 segundos, acrescido ou diminuído de um valor randômico de até 5 segundos, para diminuir a probabilidade do envio de dados pelos nós ao mesmo tempo. O tamanho do *payload* foi escolhido de forma a prevenir que o tamanho do *frame* não ultrapassasse os 127 bytes determinados pelo padrão IEEE 802.15.4, evitando assim a fragmentação em dois ou mais pacotes.

O tempo total de execução de cada simulação foi de 20 minutos, possibilitando o envio de ao menos 30 mensagens de cada nó por simulação.

Para cada cenário, a aplicação de cálculo de caminhos mínimos recebia as informações vindas dos *motes* e calculava os melhores caminhos para o nó (6LBR) onde estava o servidor

[19] https://github.com/marciolm/sd6wsn - Acesso em 10 de janeiro de 2018.

da aplicação de geração de tráfego, que por sua vez utilizava os caminhos calculados para o retorno das mensagens aos nós de origem.

A alteração no alcance de transmissão influi diretamente no número de vizinhos de cada nó, e com isso, tanto o RPL quanto a aplicação de cálculo de caminhos que utilizam como métrica de qualidade o ETX aditivo, escolhem os caminhos com menor número de saltos até o 6LBR. A Figura 6.4 ilustra a influência do alcance no número de vizinhos para cada nó. Na Figura 6.4a, para o alcance de 25 metros, cada nó possui de 3 a 6 vizinhos, aumentando para 7 a 14 vizinhos para 50 metros (Figura 6.4b), de 13 a 18 para 100 metros (Figura 6.4c) e de 19 vizinhos para todos os nós para 150 metros de alcance (Figura 6.4d). Neste último caso, todos os nós possuem conectividade direta ao 6LBR e, com isso, a RSSF inteira preferencialmente terá apenas um salto, a não ser que a qualidade de conexão direta entre os nós até o 6LBR os forcem a utilizar mais de um salto.

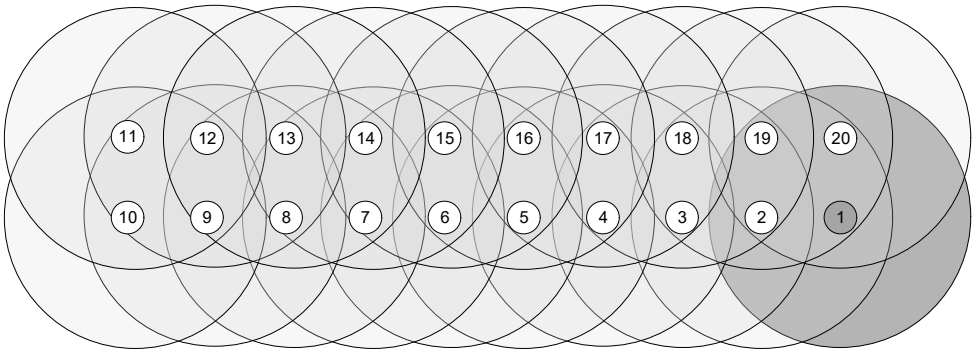

Figura 6.4a – Alcance dos *motes* de 25m

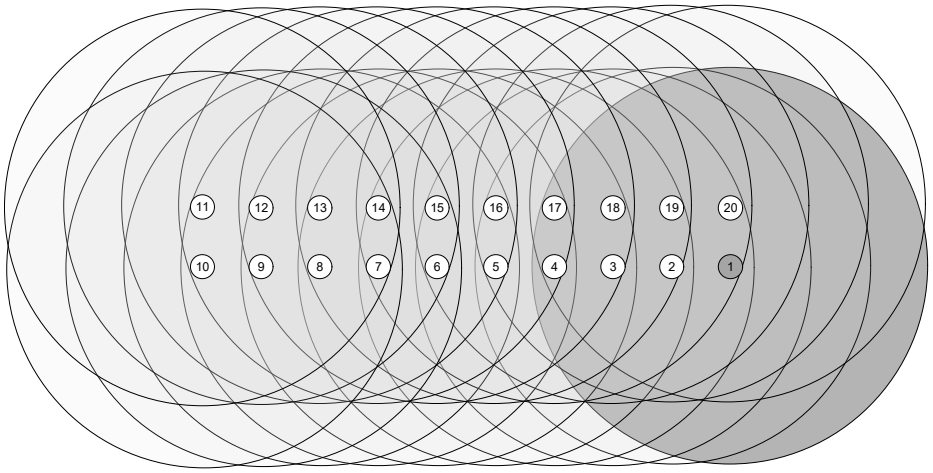

Figura 6.4b – Alcance dos *motes* de 50 m

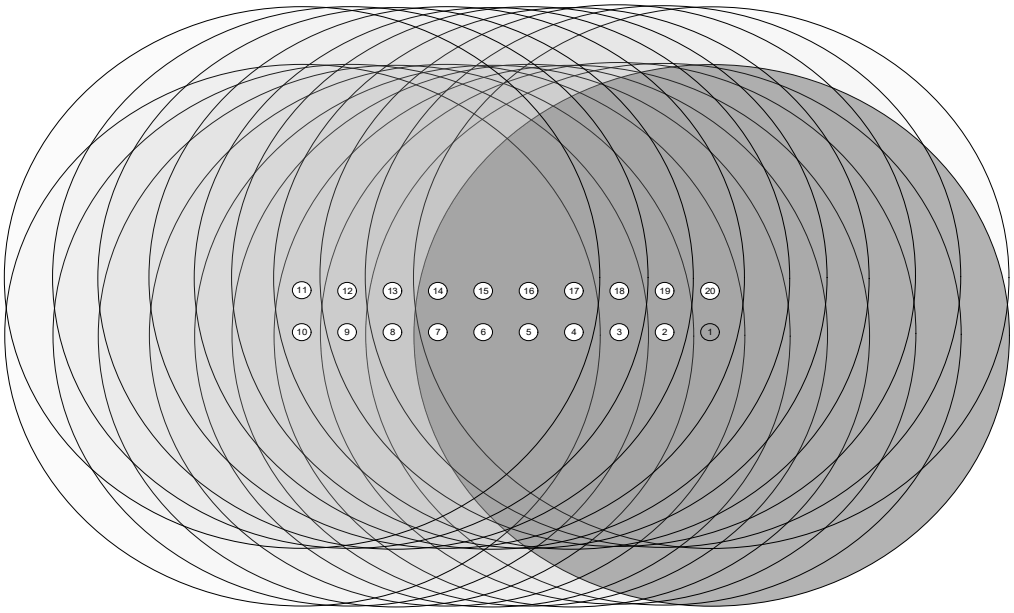

Figura 6.4c – Alcance dos *motes* de 100 m

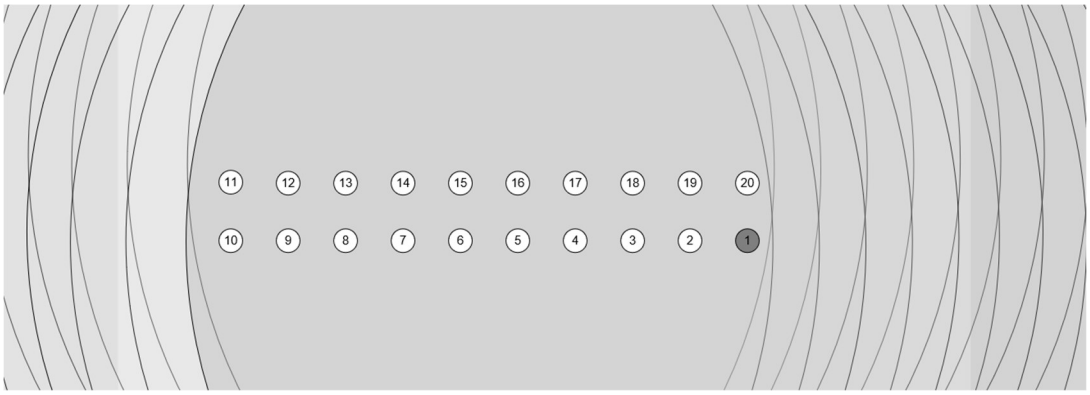

Figura 6.4d – Alcance dos *motes* de 150 m

Foram criados quatro *scripts* no COOJA para simularem os cenários, um para cada alcance. Para que fosse possível a comunicação do simulador com a máquina real Linux, onde foram executados os programas "controlador" e a aplicação de "cálculo de caminhos mínimos", as simulações rodaram em tempo real, ou seja, em 100 % da velocidade. O modo utilizado no COOJA foi o *nogui*, sem a interface gráfica, para melhoria de desempenho e para a geração dos arquivos de resultados, que continham o *timestamp* do envio e retorno das mensagens de teste, permitindo assim o cálculo da latência de ida e volta dos pacotes. O *script* também contabilizou os pacotes perdidos, informando assim o PRR total para cada cenário.

Para a comparação entre o número de pacotes SD6WSNP que foram necessários para a execução dos processos da arquitetura SD6WSN (descoberta e manutenção de topologia) e da aplicação de "cálculo de caminhos mínimos" em relação ao número de mensagens trocadas pelo RPL, foi realizada a captura de pacotes trafegados na RSSF. Essa captura é realizada através de um *sniffer* em redes reais, como o apresentado na Seção 5.3.2, mas no ambiente COOJA é realizado por um *plugin* denominado "*radiologger-headless*"[20] , que grava em um arquivo compatível com o programa de análise de protocolos "*Wireshark*" toda a atividade de transmissão e recepção de dados da simulação. Pela análise desse arquivo, foi possível mensurar o número de pacotes de cada protocolo.

Para cada cenário, cada simulação foi executada trinta vezes com o roteamento dos pacotes executado pelo RPL e trinta vezes utilizando a aplicação de "cálculo de caminhos mínimos" para a geração dos fluxos. O código do controlador e da aplicação de caminho mínimo, escritos na linguagem Node.js, também estão disponíveis no sítio desse projeto.

6.1.3. Análise dos resultados

Esta Seção apresenta os resultados obtidos nas simulações realizadas no COOJA para os cenários de rede AMI, e buscou comparar o desempenho da rede com as rotas dos pacotes definidas pelo RPL com a mesma rede, porém com as rotas definidas pela aplicação SD6WSN.

6.1.3.1. Latência média e número de saltos

O primeiro cenário analisado foi o de 25 metros de alcance, o que apresentou o maior número de saltos entre o 6LBR e o nó mais distante geograficamente, de número 11. O algoritmo Dijkstra calculou inicialmente o seguinte caminho para este nó com nove saltos:

```
['n1', 'n2', 'n3', 'n17', 'n16', 'n15', 'n14', 'n8', 'n12', 'n11']
```

E no caminho inverso, do nó para o 6LBR, o caminho calculado foi diferente, mas também com nove saltos:

```
['n11', 'n12', 'n13', 'n7', 'n6', 'n16', 'n17', 'n3', 'n2', 'n1']
```

Isto ocorre porque os caminhos são calculados com os valores medidos de ETX de cada enlace nos dois sentidos, o que não acontece no RPL, que assume que os valores de ETX são idênticos por enlace, e somente utiliza o valor de ETX dos enlaces no sentido de nó "pai" para nó "filho".

[20] https://github.com/cetic/cooja-radiologger-headless - Acesso em 10 de janeiro de 2018.

A Figura 6.5 mostra a latência média para cada nó, para a rede com rotas definidas pelo RPL e pela aplicação de “cálculo de caminhos mínimos”. A latência média foi maior para os nós mais distantes do 6LBR devido ao elevado número de saltos, explicado pelo baixo alcance de transmissão que foi definido para este cenário. Contatou-se que não houve diferença significativa (dentro do intervalo de confiança de 95 %, indicado pelas barras de erros) entre as medidas feitas para cada nós, mostrando a equivalência de desempenho das rotas definidas pela aplicação SD6WSN com o roteamento RPL.

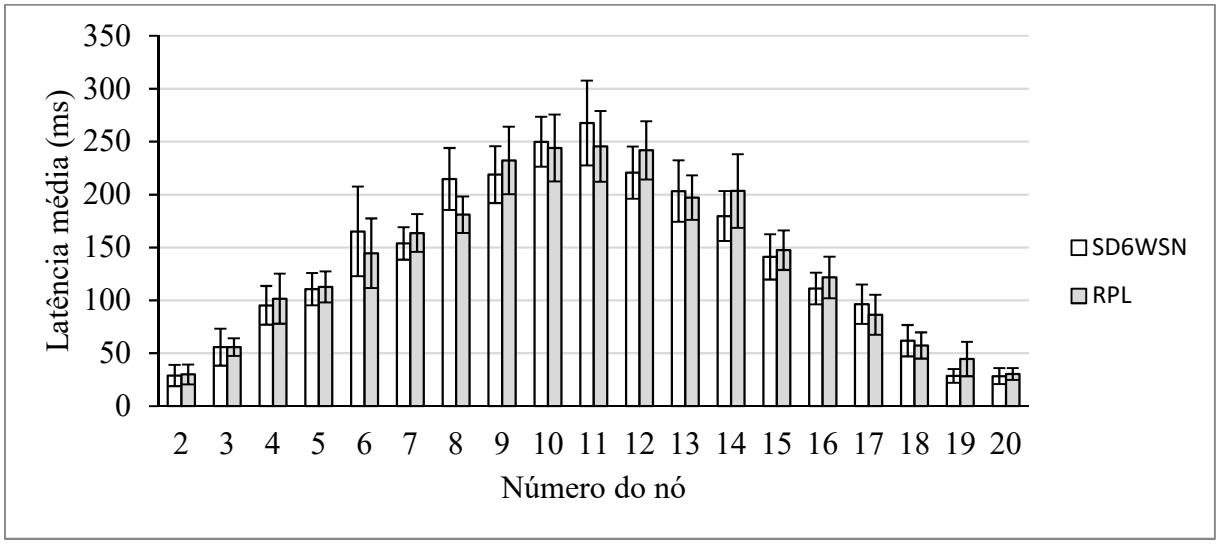

Figura 6.5 – Latência média para o cenário de 25 m de alcance

O cenário para o alcance de 50 metros apresenta caminhos mais curtos, já que não são utilizados apenas nós adjacentes para a composição dos melhores caminhos. Para o nó “11” os seguintes caminhos foram calculados pelo algoritmo Dijkstra [SED2011]:

6LBR ao nó “11”:

```
['n1', 'n17', 'n14', 'n11']
```

Nó “11” ao 6LBR:

```
['n11', 'n14', 'n17', 'n1']
```

Neste caso, os dois caminhos foram de apenas três saltos e passando pelos mesmos nós. O número de saltos menor possibilitou uma diminuição da latência média para os nós mais distantes do 6LBR (em comparação com o cenário de 25m), como observado na Figura 6.6.

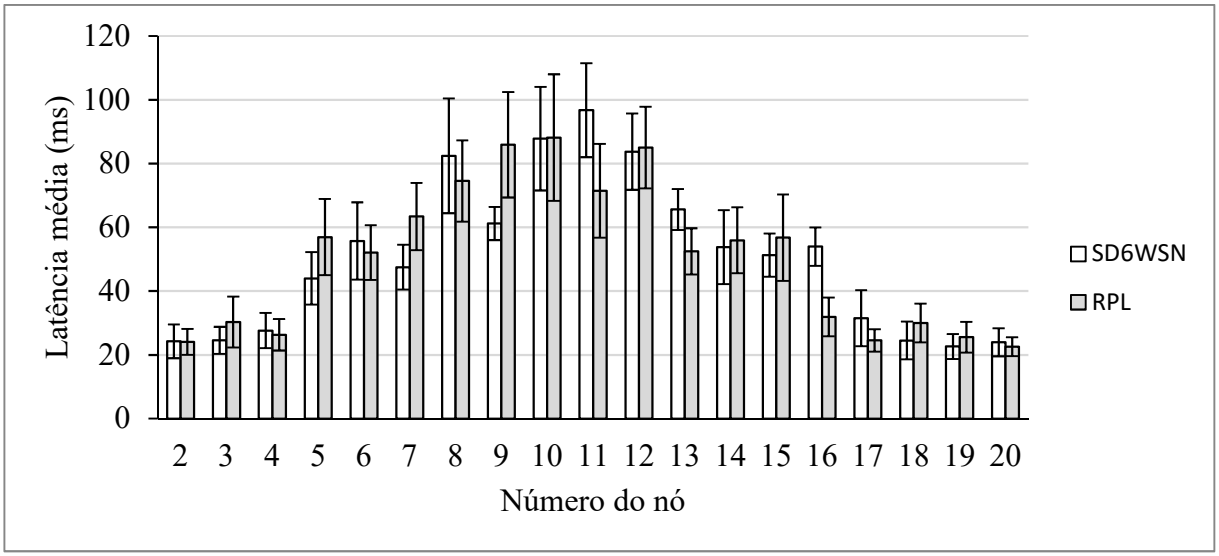

Figura 6.6 – Latência média para o cenário de 50 m de alcance

Nesta mesma Figura 6.6 pode-se observar que os resultados para os caminhos adotados pelo RPL e pelo SD6WSN apresentaram resultados similares dentro do intervalo de confiança de 95 %, com exceção dos caminhos para o nó número 9, onde a latência foi inferior para o SD6WSN e para o nó número 16, onde a latência foi inferior para o RPL. As diferenças de latência nesses dois nós se devem à diferença do cálculo de caminho no sentido do nó para o 6LBR, que caso do RPL segue o mesmo caminho do caminho do 6LBR para o nó, mas na aplicação SD6WSN é realizado para cada um dos sentidos.

Para o alcance de 100 metros, o número de saltos diminui mais ainda, com apenas dois saltos para os nós mais distantes, como mostrado na Figura 6.7. Os caminhos calculados para o nó mais distante de número “11” foram também simétricos neste cenário, com dois saltos:

6LBR ao nó “11”:

```
['n1', 'n5', 'n11']
```

Nó “11” ao 6LBR:

```
['n11', 'n5', 'n1']
```

As latências médias dos caminhos calculados pelo SD6WSN e pelo RPL também foram equivalentes, dentro do intervalo de confiança de 95 %.

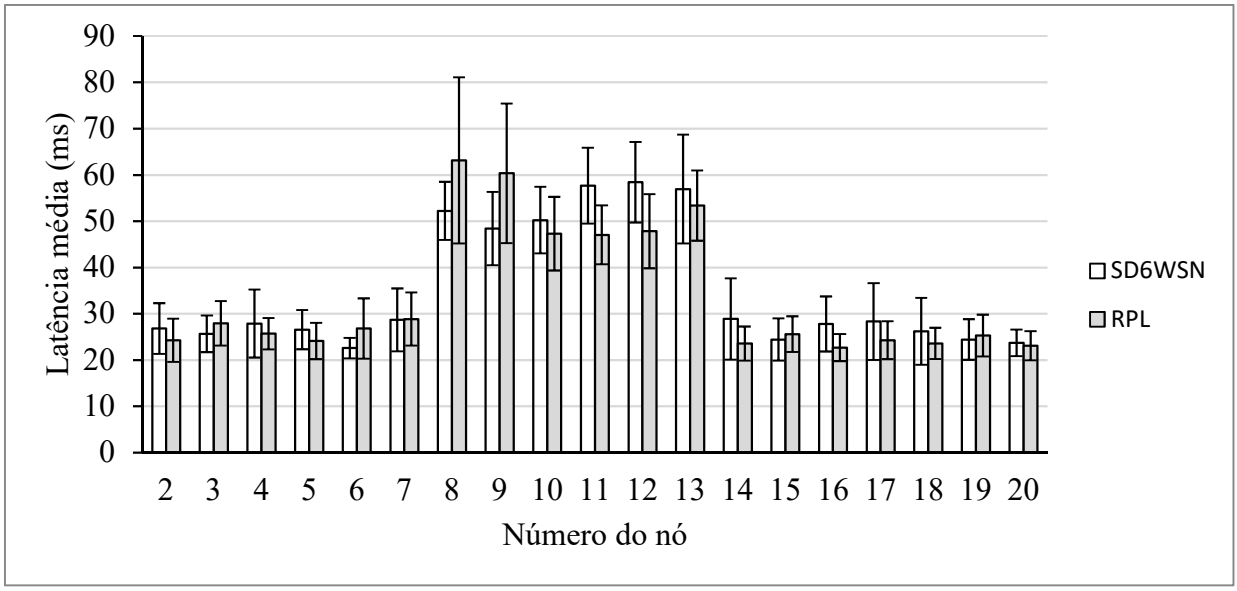

Figura 6.7 – Latência média para o cenário de 100 m de alcance

O cenário com 150 metros de alcance de transmissão para cada nó permitiu que toda a RSSF se comunicasse com o 6LBR diretamente, como em uma RSSF do tipo ponto-multiponto.

O gráfico de latência média para este cenário é apresentado na Figura 6.8, sendo uniforme para todos os nós. Não houve diferença entre as latências medidas entre os caminhos calculados pelo RPL e pela aplicação SD6WSN, dentro do intervalo de confiança de 95 %, indicado no gráfico pelas barras de erro.

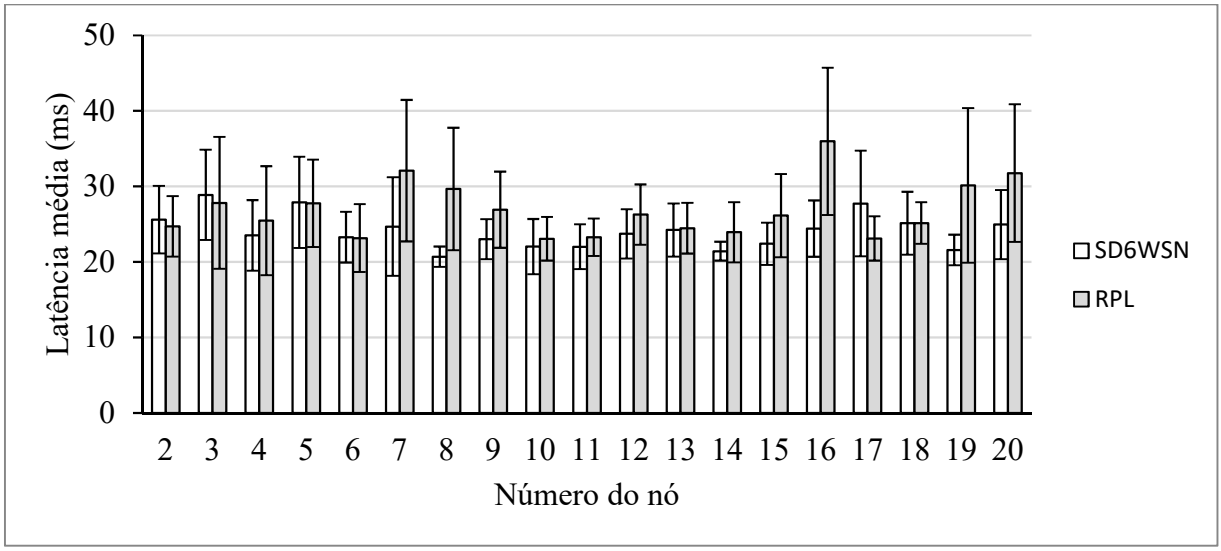

Figura 6.8 – Latência média para o cenário de 150 m de alcance

6.1.3.2. Medidas de PRR

O gráfico de taxa recepção de pacotes dos três cenários é apresentado na Figura 6.9. O cenário com menor número de saltos é o que apresenta maior taxa de recepção de pacotes, pois a probabilidade de perda é maior quando há o encaminhamento de um pacote de nó para outro, dentro da taxa de sucesso de cada transmissão definida na simulação em 75 %. O encaminhamento pelo caminho definido pela aplicação SD6WSN teve PRR superior (dentro do intervalo de confiança de 95 %) ao RPL no caso de elevado número de saltos (no cenário de 25 m) e equivalente ao RPL no cenário de um salto (alcance de 150 m), porém teve uma taxa de sucesso menor nos casos intermediários. Uma investigação posterior sobre este comportamento apontou para uma característica da implementação do RPL no Contiki, que mantém em sua tabela de nós vizinhos apenas os endereços de *nexthop* dos últimos nós que receberam mensagens do protocolo RPL. Assim, caso o nó escolhido de para o encaminhamento de um pacote pela aplicação SD6WSN não esteja na tabela do RPL, há um descarte do mesmo. A mudança do código da biblioteca RPL do Contiki, que realizará a ampliação desta tabela, será tratada em trabalhos futuros.

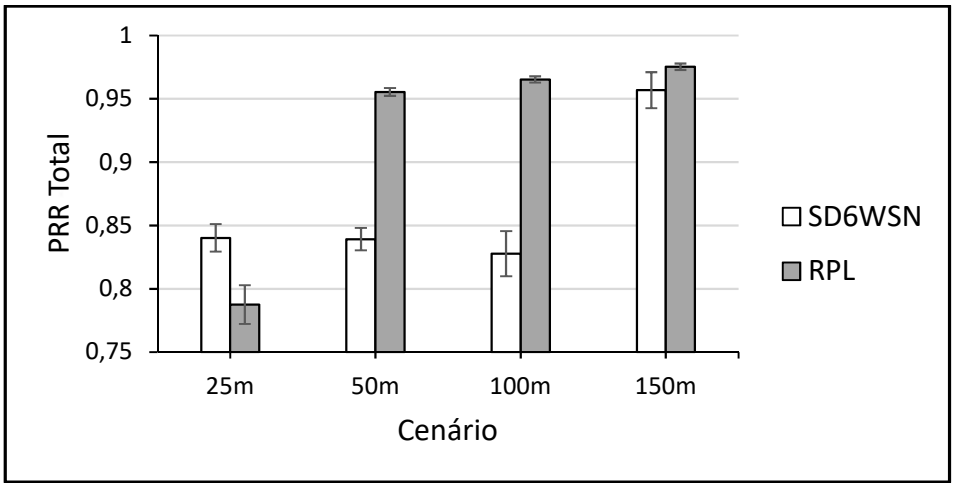

Figura 6.9 – PRR para cada cenário

6.1.3.3. Tráfego originado por mensagens SD6WSNP

Uma outra mensuração que foi considerada para análise foi o *overhead* causado pelo SD6WSNP no tráfego total da RSSF. A Tabela 6.1 compara o número de mensagens trafegadas pelo SD6WSNP em relação ao RPL, mesmo estando este sempre presente nos dois casos.

Tabela 6.1 – Percentual de mensagens SD6WSNP em relação às mensagens RPL

Cenário	Número de mensagens RPL	Número de mensagens SD6WSNP	Percentual de mensagens SD6WSNP em relação às mensagens RPL (%)
25 m	30412	1736	5,71
50 m	13245	239	1,80
100 m	9186	160	1,74
150 m	6771	73	1,08

Pode-se notar que a quantidade de mensagens diminui muito no cenário de 150 metros, com apenas um salto. Isto se deve ao menor número de entradas na tabela de fluxos nos nós, de apenas uma, relativa ao encaminhamento ao 6LBR, o qual recebe 19 entradas na tabela de fluxos, uma para cada nó de encaminhamento. A Tabela 6.2 mostra o número total de mensagens enviadas para a instalação de entradas na tabela de fluxos (*flow-mod insert*) de todos os nós e para o 6LBR, para cada cenário desta determinada topologia.

Tabela 6.2 – Número de mensagens SD6WSNP *flow-mod insert* por cenário

Cenário	**Número de mensagens SD6WSNP *flow-mod insert***
25m	182
50m	76
100m	44
150m	38

6.1.4. Conclusão

Este capítulo descreveu os ensaios de comprovação do funcionamento do *framework* proposto com a utilização da aplicação de "cálculo de caminhos mínimos" para a determinação das rotas entre nós e o 6LBR, considerada padrão para as RSSFs. Esta aplicação não difere com o objetivo do RPL, apesar da aplicação calcular os melhores caminhos nos dois sentidos sem assumir que eles são iguais nos dois sentidos para todos os enlaces, como ocorre no RPL.

Foi constatado um comportamento muito semelhante da aplicação de "cálculo de caminhos mínimos" com o RPL na formação dos caminhos em todos os cenários, no aspecto da determinação do número de saltos com o aumento do número de vizinhos comuns para os nós, por ser utilizada a métrica ETX como sendo o custo do enlace em ambos os casos.

Os resultados dos experimentos não mostraram uma variação significativa para esta aplicação que utilizou o algoritmo Dijkstra, porém a capacidade que o *framework* tem de modelar os caminhos dos pacotes de dados dentro da RSSF de acordo com qualquer algoritmo que for definido, traz uma perspectiva de criação de aplicações específicas para redes que não tenham somente um ponto de convergência de tráfego, como em redes *peer-to-peer*, onde o tráfego entre nós pode ser otimizado em relação ao possível com RPL. O estudo do desempenho de comunicação do tipo *peer-to-peer* entre nós da mesma RSSF é abordado na Seção 6.2.

6.2. Medida de desempenho da comunicação entre os nós de uma RSSF

O propósito deste ensaio é de mensurar a latência da comunicação entre os nós de uma RSSF com os caminhos entre eles definidos por uma aplicação SD6WSN, em comparação ao obtido com o roteamento RPL. O RPL, pela utilização de um DODAG, otimiza a comunicação entre os nós e o 6LBR, mas o caminho dos pacotes entre os nós depende da posição escolhida pelo RPL dentro do DODAG. No caso de nós pertencentes ao mesmo ramo, se um nó for "pai" ou "filho" de outro, a comunicação é direta, mas caso contrário, haverá um ou mais saltos entre eles até um nó ancestral comum.

A aplicação SD6WSN de "cálculo de caminhos entre nós da RSSF" não utiliza o conceito de DODAG, e calcula, por meio do algoritmo Dijkstra o melhor caminho entre os nós de origem e destino, instalando entradas que direcionam o tráfego de uma forma otimizada na tabela de fluxos dos nós pertencentes ao caminho calculado.

6.2.1. Metodologia

Para esse ensaio foi construído um cenário de simulação constituído por 25 nós de sensoriamento em topologia do tipo grade 5x5 (Figura 6.10), mais o 6LBR (nó de número "1"), necessário para prover o roteamento RPL do plano de controle SD6WSN e para formar a rede RPL para as medidas de latência comparativas com as obtidas com a aplicação de cálculo de melhores caminhos entre os nós da RSSF.

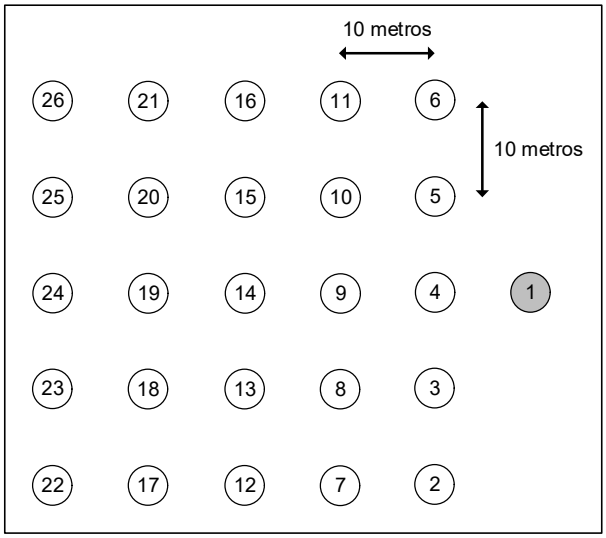

Figura 6.10 – Topologia grade 5x5

O número de combinações possíveis de tráfego entre clientes e servidores é dada pela equação:

```
número de pares = número de nós x (número de nós – 1)
```

resultando em 600 combinações de origens e destinos para este cenário. Todos os nós tiveram seus *firmwares* compilados com as funções de servidor e cliente, para poderem enviar e receber pacotes de qualquer nó da RSSF.

Foi determinada para o ensaio a medida de latência de 10 % dessas combinações escolhidas de forma aleatória, em 3 grupos de 20 pares por simulação, onde não se repetia em cada grupo os nós de origem, mas com os nós de destino escolhidos livremente (exceto quando coincidia ser o mesmo nó de origem), inclusive com repetição, com outra origem. Cada nó de origem enviou 30 pacotes de 20 bytes de *payload*, sendo um a cada 10 segundos, para o destino determinado no início de cada rodada. O tamanho do *payload* foi escolhido de forma que não houvesse a fragmentação do pacote em dois ou mais *frames* IEEE 802.15.4.

Para a mensuração da latência, foi realizado o cálculo das diferenças entre os *timestamps* de envio dos pacotes pelos nós de origem e das recepções pelos nós de destino.

6.2.2. Ambiente de simulação

O cenário de simulação no ambiente COOJA é mostrado na Figura 6.11 e é similar ao utilizado na Seção 6.1, mas a aplicação utilizada neste caso foi a de “cálculo de caminhos entre

os nós de uma RSSF", desenvolvida especialmente para calcular, pelo algoritmo Dijkstra, as melhores rotas entre os nós de origem e destino previamente sorteados.

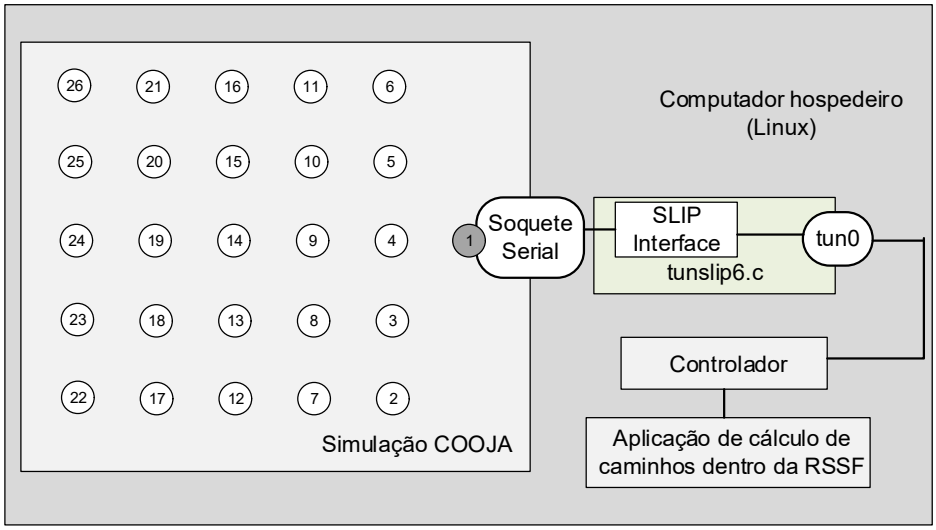

Figura 6.11 – Integração entre a simulação COOJA e o controlador SD6WSN

Após a definição dos nós de origem e destino para cada rodada de 20 pares, os processos de "descoberta e manutenção de topologia" descritos na Seção 4.4.2 realizam a leitura dos parâmetros de qualidade entre os nós vizinhos e enviam essas informações para a aplicação, que faz o cálculo dos caminhos e os instala (por intermédio do controlador) em todos os nós pertencentes aos caminho encolhido para a comunicação entre cada par de nós. No caso de alterações das métricas de qualidade, é realizado o recálculo e consequente alteração na tabela de fluxos de todos os nós envolvidos em determinada rota.

O sorteio dos pares de nós e a definição do nó de destino dos pacotes enviados pelo processo cliente instalado em todos os nós, é realizado em seguida por um programa auxiliar escrito especificamente para este ensaio, também disponível no sítio Github[21]. Para a definição dos nós de destino, mensagens CoAP do tipo PUT, com a URI:

```
coap://<IPv6 do nó origem>/test/udptest?index=<IPv6 de destino>
```

são enviadas para cada um dos 20 nós de origem sorteados.

A simulação COOJA foi executada no modo *nogui* com velocidade de 100 %, modelo UDGM, com 25 metros de alcance e 50 metros de interferência, de tal forma a limitar o alcance

[21] https://github.com/marciolm/sd6wsn/blob/master/scripts/sd6wsn-25nodes-app-v2.js

de cada nó aos seus vizinhos imediatos. As taxas de sucesso de transmissão e de recepção adotadas foram de 75 % e de 100 %, respectivamente.

Foram executadas 3 rodadas com o envio de 30 pacotes UDP por cada um dos 20 nós de origem dos pares de cada uma das 10 simulações, com a semente aleatória do modelo UDGM alterada à cada simulação.

Para possibilitar a comparação entre os resultados obtidos pelo uso da arquitetura SD6WSN com os obtidos pelo roteamento RPL, foi executado o mesmo número de simulações nos mesmos pares de nós apenas com o roteamento RPL.

6.2.3. Análise dos resultados

As medidas de latência foram realizadas pela diferença entre os *timestamps* registrados no momento do envio dos pacotes pelos nós de origem e na chegada aos nós de destino. Conforme o sorteio dos pares, um nó poderia receber pacotes de diversas origens, que foram identificadas pelo conteúdo das mensagens enviadas.

A Figura 6.12 mostra as latências observadas em duas rodadas (uma para o SD6WSN e outra para o RPL), compostas por 20 pares sorteados de origem e destino de tráfego, com o intervalo de confiança de 95 %, obtido por 30 medidas por cada par de nós, indicado pelas barras de erro.

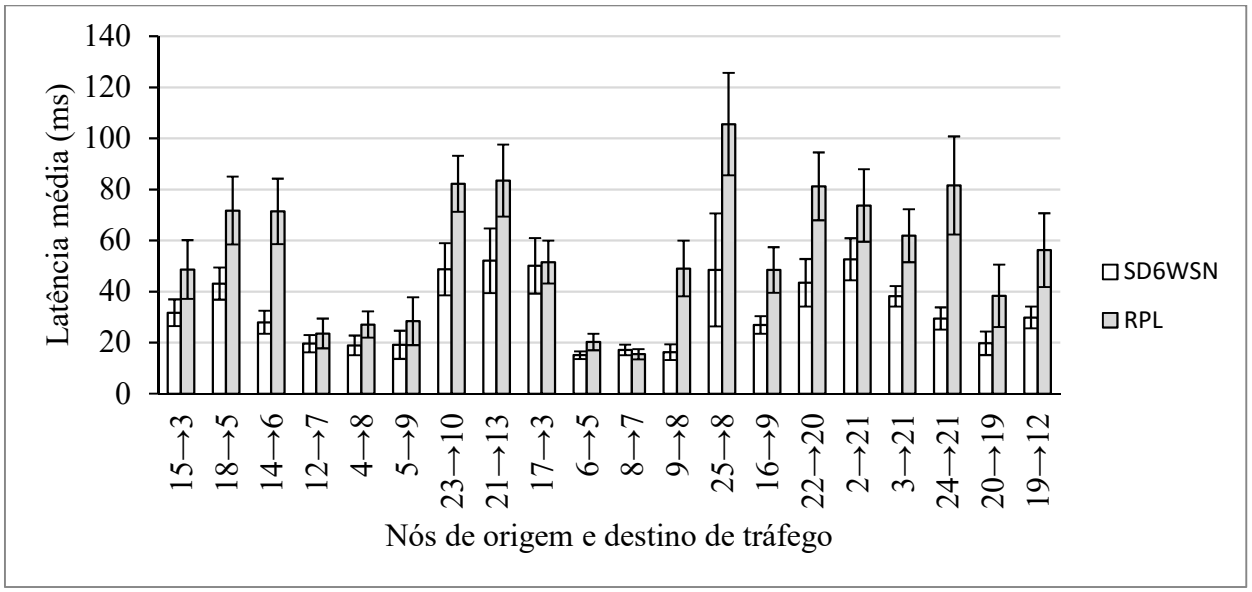

Figura 6.12 – Latência de cada par de nós de origem e destino

A latência medida entre os nós foi menor ou de mesmo valor (dentro do intervalo de confiança de 95 %) para todos os caminhos calculados pela aplicação SD6WSN de todos os pares dessa rodada. Em alguns casos de nós fisicamente adjacentes, como os dos pares 12→7, 4→8, 5→9, 6→5, 8→7, os caminhos escolhidos pelo SD6WSN e pelo RPL apresentaram latências similares, porém em casos como os dos pares 9→8 e 20→19, a latência do caminho RPL foi superior à observada para o caminho determinado SD6WSN, denotando a passagem dos pacotes por um ou mais nós intermediários, ou pais preferenciais, comuns aos dois nós na montagem do DODAG RPL.

A Figura 6.13, obtida com o uso da ferramenta de visualização de redes RPL Foren6[22] a partir de capturas de pacotes realizadas no formato *Wireshark*, apresenta a configuração do DODAG criado em determinado momento, mostrando os caminhos escolhidos pelo RPL entre os nós do cenário da Figura 6.10.

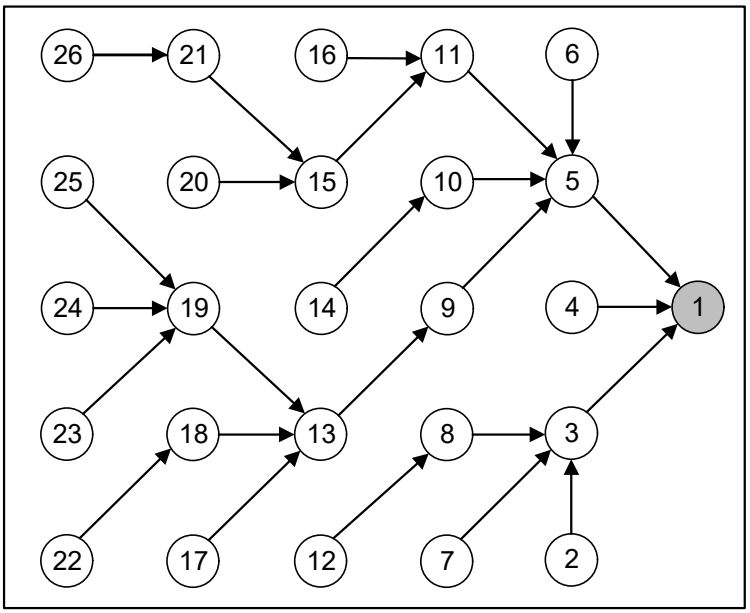

Figura 6.13– Representação das rotas definidas pelo RPL

Para fins de ilustração, a Figura 6.14 mostra os caminhos escolhidos pela aplicação SD6WSN e pelo RPL para o par de nós 25 e 8 no ensaio realizado, que apresentou a maior latência dos resultados que foram apresentados na Figura 6.12. Pode-se observar que a rota definida pela aplicação SD6WSN (representada por setas tracejadas) possuía apenas três saltos,

[22] http://cetic.github.io/foren6/index.html - Acesso em 10 de janeiro de 2018

enquanto que a rota RPL (representada por setas sólidas) possuía sete saltos, explicando assim a maior latência para essa rota.

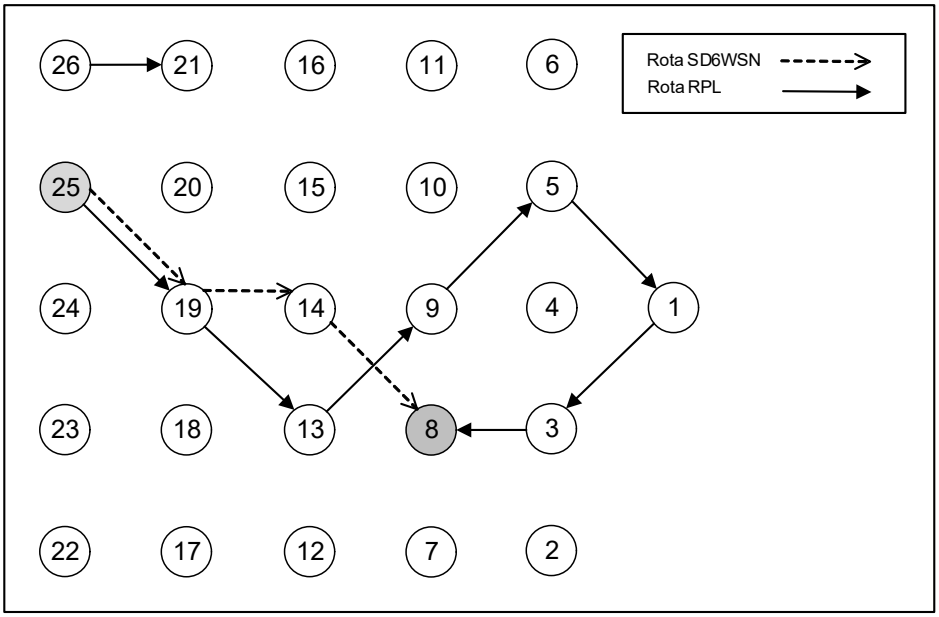

Figura 6.14– Rotas entre os nós 25 e 8 utilizando-se o SD6WSN e o RPL

A menor latência média para os caminhos calculados pela aplicação SD6WSN se repetiu em todas as 30 rodadas de 20 pares, o que pode ser verificado na Figura 6.15, onde é apresentada a latência média de todas as 10 simulações, com o intervalo de confiança de 95 % sendo representado pelas barras de erros.

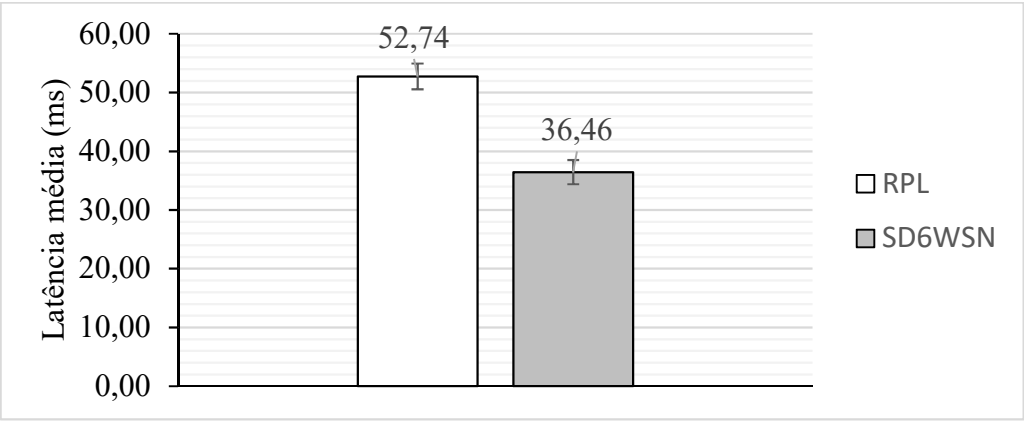

Figura 6.15 – Latência média da RSSF para as rotas RPL e SD6WSN

6.2.4. Conclusão

O RPL, por empregar um DODAG com raiz no 6LBR, é otimizado para o tráfego entre os nós e a raiz. Como a aplicação SD6WSN calcula o caminho a partir da raiz situada no nó de origem, observou-se uma diminuição na latência na comunicação entre nós vizinhos e não vizinhos, que pelo RPL utilizaram um nó ancestral comum entre os nós de origem e destino. A otimização dos caminhos provida pela aplicação SD6WSN proporcionou uma diminuição da latência calculada pela média das latências de todos os pares aleatórios de nós envolvidos na simulação em 30,87 %, da latência observada para os mesmos pares com a utilização do roteamento RPL.

Conclusão

Neste trabalho foi proposta uma abordagem SDN para RSSFs de múltiplos saltos do tipo 6LoWPAN e que utilizam o RPL como protocolo de roteamento. Na concepção do *framework* SD6WSN, foram consideradas as características específicas das RSSFs, como baixa taxa de transferência de dados, elevada latência, possibilidade de perda de pacotes e baixo poder de processamento dos dispositivos integrantes. Buscou-se também o aproveitamento da flexibilidade proporcionada pelo roteamento por fluxos, que tendo como base as características de cada pacote ingressante na RSSF, permite o desenvolvimento de aplicações mais específicas, voltadas ao conteúdo dos dados que trafegam pela rede e assim tratá-los de forma diferente dos demais.

Durante o transcorrer do projeto, percebeu-se o desafio de se desenvolver um código otimizado para dispositivos de recursos limitados como os empregados em RSSFs, onde a capacidade de RAM, de poucas dezenas de kbytes, é o maior limitante. O sistema operacional de código livre Contiki, que serviu de base para a implementação do projeto, conseguiu prover com seu ambiente de desenvolvimento maduro e bem documentado, as ferramentas necessárias para a implementação do agente SD6WSN, inclusive o código do servidor CoAP, que foi fundamental para a escrita de um dos componentes principais da arquitetura, o agente SD6WSN.

Antes da proposição do *framework*, foi realizado um estudo sobre o funcionamento e limitações do protocolo RPL e como o paradigma SDN, que estava sendo adotado em outros campos da área de redes de computadores, poderia melhorar o desempenho das RSSFs nos quesitos de distribuição e gerenciamento de tráfego, além de proporcionar uma visão sistêmica da rede.

Para a validação da abordagem SDN em redes 6LoWPAN, foi proposto um *framework* composto por duas partes, a primeira sendo a arquitetura, onde foram definidos os componentes integrantes da solução, a comunicação entre eles e os processos de estabelecimento e manutenção da topologia definida por *software* para cada fluxo. A segunda parte tratou da definição do protocolo *Southbound* do plano de controle da SDN, denominado SD6WSNP, o

qual foi inspirado em um protocolo idealizado para redes cabeadas, o *OpenFlow*. Neste protocolo foram implementadas todas as mensagens necessárias para a obtenção dos parâmetros dos nós e para a instalação de entradas as tabelas de fluxo, que contêm as informações para identificação dos fluxos e as ações para cada um deles, o que comanda o comportamento do tráfego dentro da RSSF. Por meio do protocolo SD6WSNP, um componente de *software* denominado "controlador" provê toda a comunicação para a leitura de parâmetros e instalação de regras nos nós, além da execução dos processos de descoberta e manutenção de topologia e a comunicação com as aplicações, que definem o comportamento dos pacotes na rede.

Dentre as aplicações possíveis de serem implementadas para a execução dos experimentos da prova de conceito, foram escritas duas aplicações, uma que utiliza o algoritmo Dijkstra para o cálculo dos menores caminhos entre os nós e o 6LBR e outra que utiliza o mesmo algoritmo para o cálculo dos melhores caminhos entre pares de nós da RSSF.

Na etapa de ensaios, primeiramente foi criado um cenário típico de redes AMI dentro do simulador COOJA onde foram executados ensaios para quatro configurações distintas de meio de propagação de rádio, para que o alcance de transmissão fosse modificado de forma a alterar o número de saltos entre nós. Para fins de validação, os dados obtidos pelas simulações foram comparados com os obtidos por outra série de simulações onde somente o protocolo RPL foi executado. Para esta aplicação, os resultados foram equivalentes aos obtidos pelo RPL, por utilizarem o mesmo algoritmo para a determinação de melhores caminhos.

No segundo ensaio de simulação foi utilizada uma topologia do tipo grade e alcance fixo, para que pudesse ser comparada a latência média da comunicação de pares de nós escolhidos aleatoriamente e caminhos definidos pela aplicação SD6WSN com os mesmos nós, mas com os caminhos definidos pelo RPL. Neste cenário, a média de latência de comunicação entre os pares sorteados apresentou significativa redução em relação às rotas calculadas pelo RPL. Esta melhoria de desempenho se deve à execução do algoritmo Djikstra para todos os pares envolvidos na simulação e não somente para a montagem do DODAG, evitando-se assim o tráfego por nós ancestrais comuns aos dois nós, como ocorre no roteamento RPL.

Para trabalhos futuros com o emprego do *framework* proposto, pode-se citar a implantação do algoritmo de planejamento de potência em redes AMI, que foi tema de um artigo [PEN2017] publicado durante o transcorrer desse trabalho, aplicações de balanceamento de carga de enlaces, priorização de tráfego por tipo de dado, virtualização de redes, além de estudos para a aplicação de outras métricas para a determinação de qualidade dos enlaces.

Referências Bibliográficas

[ALI2012] ALI, H. *A Performance Evaluation of RPL in Contiki*. Dissertação de Mestrado em ciência da computação. Swedish Institute of Computer Science, 2012.

[ANC2012] ANCILLOTTI, E.; BRUNO, R.; CONTI, M. *RPL routing protocol in advanced metering infrastructures*: *An analysis of the unreliability problems*. In: SUSTAINABLE INTERNET AND ICT FOR SUSTAINABILITY (SUSTAINIT), October 2012. IEEE, 2012, p. 1-10.

[ANC2013] ANCILLOTTI, E.; BRUNO, R.; CONTI, M. *The role of the RPL routing protocol for smart grid communications*. IEEE Communications Magazine, 2013, p. 75-83.

[BEN2013] BENTON, K.; CAMP, L.; SMALL, C. *Openflow vulnerability assessment*. In: PROCEEDINGS OF THE SECOND ACM SIGCOMM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, August 2013. ACM, 203, p. 151-152.

[CLA2011] CLAUSEN, T.; HERBERG, U.; PHILIPP, M. *A critical evaluation of the ipv6 routing protocol for low power and lossy networks (RPL)*. In: WIRELESS AND MOBILE COMPUTING, NETWORKING AND COMMUNICATIONS (WIMOB), 7th International Conference, 2011. IEEE, 2011, p. 365-372.

[COS2012] COSTANZO, S.; GALLUCCIO L.; MORABITO, G.; PALAZZO, S. *Software defined wireless networks*: *Unbridling sdns*. In: PROCEEDINGS OF THE 2012 EUROPEAN WORKSHOP ON SOFTWARE DEFINED NETWORKING - EWSDN, 12, 2012. IEEE, 2012, p. 1–6.

[CUL2004] CULLER D.; ESTRIN D.; SRIVASTAVA M. *Guest Editors Introduction: Overview of Sensor Networks.* In: COMPUTER, August 2004. vol. 37, no. 8, p. 41-49.

[DAW2012] DAWANS, S.; DUQUENNOY, S.; BONAVENTURE, O. *On link estimation in dense RPL deployments*. In: LOCAL COMPUTER NETWORKS WORKSHOPS (LCN WORKSHOPS), 37th Conference, October 2012. IEEE, 2012. p. 952-955.

[DUN2004] DUNKELS, A.; GRÖNVALL, B.; VOIGT, T. *Contiki-a lightweight and flexible operating system for tiny networked sensors*. In: LOCAL COMPUTER NETWORKS, 29th Annual IEEE International Conference, November 2004. IEEE, 2004, p. 455-462.

[GAD2012] GADDOUR, O.; KOUBAA, A.; CHAUDHRY, S.; TEZEGHDANTI, M.; CHAARI, R.; ABID, M. *Simulation and performance evaluation of DAG construction with RPL*. In: COMMUNICATIONS AND NETWORKING (COMNET), Third International Conference, March 2012. IEEE, 2012, p. 1-8.

[GAL2015] GALLUCCIO L.; MILARDO S.; MORABITO G.; PALAZZO S.; *SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks*. In: 2015 IEEE CONFERENCE ON COMPUTER COMMUNICATIONS (INFOCOM). Kowloon, IEEE, 2015, p. 513-521.

[HAM2009] HAMILTON, J. *Networking*: The last bastion of mainframe computing. 2009. Disponível em: http://perspectives.mvdirona.com/2009/12/networking-the-last-bastion-of-mainframe-computing. Acesso em 04 jan. 2018.

[HAR2017] HARADA, H.; MIZUTANI, K.; FUJIWARA, J.; MOCHIZUKI, K.; OBATA, K.; OKUMURA, R. *IEEE 802.15.4g based Wi-SUN communication systems*. IEICE Transactions on Communications, vol. 100, n. 7, 2017, p. 1032-1043.

[HER2011] HERBERG, U.; CLAUSEN, T. *A comparative performance study of the routing protocols LOAD and RPL with bi-directional traffic in low-power and lossy networks (LLN)*. In: PROCEEDINGS OF THE 8TH ACM SYMPOSIUM ON PERFORMANCE

EVALUATION OF WIRELESS AD HOC, SENSOR, AND UBIQUITOUS NETWORKS, November 2011, p. 73-80.

[IEEE2003] IEEE. *IEEE Std. 802.15.4-2003*: *IEEE Standard for Information Technology Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks -Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE, October, 2003.

[IEEE2012] IEEE. *IEEE Std. 802.15.4e-2012*: *IEEE Standard for Local and Metropolitan Area Networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LRWPANs) Amendment 1: MAC Sublayer.* IEEE, April, 2012.

[IPSO2011] VASSEUR, J.; AGARWAL, N.; HUI, J.; SHELBY, Z.; BERTRAND, P.; CHAUVENET, C. *RPL*: *The IP routing protocol designed for low power and lossy networks*. Internet Protocol for Smart Objects (IPSO) Alliance, 2011.

[KAT2013] KATHURIA, V.; MOHANASUNDARAM, G.; DAS, S. R. *A simulation study of routing protocols for smart meter networks*. In: SMART GRID COMMUNICATIONS (SMARTGRIDCOMM), October 2013. IEEE International Conference, 2013, p. 384-389.

[KIM2015] KIM, J.; FILALI, F.; KO, Y. *A lightweight CoAP-based software defined networking for resource constrained AMI devices*. In: 2015 IEEE INTERNATIONAL CONFERENCE ON SMART GRID COMMUNICATIONS (SMARTGRIDCOMM), November 2015. IEEE, 2015, p. 719-724.

[KOV2011] KOVATSCH, M.; DUQUENNOY, S.; DUNKELS, A. *A low-power CoAP for Contiki*. In: MOBILE ADHOC AND SENSOR SYSTEMS (MASS), IEEE 8th International Conference, October 2011. IEEE, 2011, p. 855-860.

[KUZ2014] KUZLU, M.; PIPATTANASOMPORN, M.; RAHMAN, S. *Communication network requirements for major smart grid applications in HAN, NAN and WAN*. Computer Networks, n. 67, 2014, p. 74-88.

[LON2012] LONG, N. T.; DE CARO, N.; COLITTI, W.; TOUHAFI, A.; STEENHAUT, K. *Comparative performance study of RPL in wireless sensor networks*. In: COMMUNICATIONS AND VEHICULAR TECHNOLOGY IN THE BENELUX (SCVT), IEEE 19th Symposium, November 2012. IEEE, 2012, p. 1-6.

[LUO2012] LUO T.; TAN, H.; QUEK, T., *Sensor OpenFlow: Enabling software-defined wireless sensor networks*. IEEE Communications Letters, vol. 16, n. 11, 2012, p. 1896-1899.

[MAH2011] MAHMUD, A; RAHMANI R. *Exploitation of OpenFlow in wireless sensor networks*. In: COMPUTER SCIENCE AND NETWORK TECHNOLOGY (ICCSNT), International Conference, 2011. ICCSNT, vol. 1, 2011, p. 594–600.

[MAZ2013] MAZZER, Y.; TOURANCHEAU, B. *Comparisons of 6LoWPAN Implementations on Wireless Sensor Networks*. In: PROC. SENSORCOMM, 2009. SENSORCOMM, 2009, p. 689–692.

[MCK2008] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON L.; REXFORD, J.; SHENKER, S.; TURNER, J. *Openflow: enabling innovation in campus networks*. SIGCOMM Comput, vol. 38, n. 2, 2008, p. 69–74.

[ONF2010] OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification 1.0*. Disponível em https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. Acesso em 10 jan. 2018.

[ONF2013] OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification 1.3.* Disponível em https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf. Acesso em 10 jan. 2018.

[OLS2014] OLSSON, J. *6LoWPAN Demistified*. Disponível em: http://www.ti.com/lit/wp/swry013/swry013.pdf. Acesso em 10 jan. 2018.

[OLI2015] OLIVEIRA, B. T.; ALVES, R.; MARGI, C. *Tinysdn: enabling multiple controllers for software-defined wireless sensor networks*. Latin America Transactions, IEEE (Revista IEEE America Latina), vol. 13, n. 11, 2015, p. 3690-3696.

[OLI2015b] OLIVEIRA, B.; ALVES, R.; MARGI, C. *Software-defined Wireless Sensor Networks and Internet of Things standardization synergism.* In: STANDARDS FOR COMMUNICATIONS AND NETWORKING (CSCN), IEEE Conference, 2015. IEEE, 2015, p. 60-65.

[OST2006] OSTERLIND, F.; DUNKELS, A.; ERIKSSON, J.; FINNE, N.; VOIGT, T. *Cross-level sensor network simulation with cooja*. In: LOCAL COMPUTER NETWORKS, PROCEEDINGS, 31st IEEE conference, November 2006. IEEE, 2006, p. 641-648.

[PAL2013] PALATTELLA, M.; ACCETTURA, N.; VILAJOSANA, X.; WATTEYNE, T.; GRIECO, L.; BOGGIA, G.; DOHLER, M. *Standardized protocol stack for the internet of (important) things*. Communications Surveys & Tutorials, IEEE, vol. 15, n. 3, 2013, p. 1389-1406.

[PED2013] PEDIREDLA, B.; WANG, K.; SALCIC, Z.; IVOGHLIAN, A. *A 6LoWPAN implementation for memory constrained and power efficient wireless sensor nodes*. In: PROC. IEEE IECON, 2013. IEEE, 2013, p. 4432–4437.

[PEN2017] PENNA, M.; MIGUEL, M.; JAMHOUR, E.; PELLENZ, M. *A power planning algorithm based on RPL for AMI wireless sensor networks*. SENSORS. v. 17, 2017, p. 679-696.

[RFC2464] CRAWFORD, M. *RFC 2464*: *Transmission of IPv6 Packets over Ethernet Networks*. Internet Engineering Task Force (IETF), December 1998. Disponível em: http://www.ietf.org/rfc/rfc2464.txt. Acesso em 10 jan. 2018.

[RFC3819] KARN, P.; BORMANN, C.; FAIRHURST, G.; GROSSMAN, D.; LUDWIG, R.; MAHDAVI, J.; MONTENEGRO, G.; TOUCH, J., WOOD, L. *RFC 3819*: *Advice for Internet Subnetwork Designers*. Internet Engineering Task Force (IETF), July 2004. Disponível em: http://www.ietf.org/rfc/rfc3819.txt. Acesso em 10 jan. 2018.

[RFC3986] BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *IETF RFC 3986*: *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force (IETF), January 2005. Disponível em: http://www.ietf.org/rfc/rfc3986.txt, January 2005. Acesso em 10 jan. 2018.

[RFC4443] CONTA, A.; DEERING, S.; GUPTA, M. *RFC 4443*: *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. Internet Engineering Task Force (IETF), March 2006. Disponível em: http://www.ietf.org/rfc/rfc4443.txt. Acesso em 10 jan. 2018.

[RFC4627] CROCKFORD, D. *RFC 4627*: *The application/json media type for JavaScript Object Notation*. Internet Engineering Task Force (IETF), July 2006. Disponível em: http://www.ietf.org/rfc/rfc4627.txt. Acesso em 10 jan. 2018.

[RFC4919] KUSHALNAGAR, N.; MONTENEGRO, G.; SCHUMACHER, C. *RFC 4919: IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals*. Internet Engineering Task Force (IETF), August 2007. Disponível em: http://www.ietf.org/rfc/rfc4919.txt. Acesso em 10 jan. 2018.

[RFC4944] MONTENEGRO, G.; KUSHALNAGAR, N.; HUI, J.; CULLER, D. *RFC 4944: Transmission of IPv6 packets over IEEE 802.15.4 networks*. Internet Engineering Task

Force (IETF), September 2007. Disponível em: http://www.ietf.org/rfc/rfc4944.txt. Acesso em 10 jan. 2018.

[RFC5867] MARTOCCI, J.; MIL, P. D.; RIOU, N.; VERMEYLEN, W. *RFC5867: Building automation routing requirements in low-power and lossy networks*. Internet Engineering Task Force (IETF), June 2010. Disponível em: http://www.ietf.org/rfc/rfc5867.txt. Acesso em 10 jan. 2018.

[RFC5826] BRANDT, A.; BURON, J.; PORCU, G. *RFC 5826*: *Home Automation Routing Requirements in Low-Power and Lossy Networks*. Internet Engineering Task Force (IETF), April 2010. Disponível em: http://ietf.org/rfc/rfc5826.txt. Acesso em 10 jan. 2018.

[RFC5673] PISTER, K.; THUBERT, P.; DWARS, S.; PHINNEY, T. *RFC 5673*: *Industrial Routing Requirements in Low-Power and Lossy Networks*. Internet Engineering Task Force (IETF), October 2009. Disponível em: http://ietf.org/rfc/rfc5673.txt. Acesso em 10 jan. 2018.

[RFC5548] DOHLER, M.; BARTHEL, D.; WATTEYNE, T.; WINTER, T. *RFC 5548*: *Routing requirements for urban low-power and lossy networks*. Internet Engineering Task Force (IETF), March 2009. Disponível em: http://ietf.org/rfc/rfc5548.txt. Acesso em 10 jan. 2018.

[RFC6282] HUI, J.; THUBERT, P. *RFC 6282*: *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. Internet Engineering Task Force (IETF), September 2011. Disponível em: http://ietf.org/rfc/rfc6282.txt. Acesso em 10 jan. 2018.

[RFC6206] LEVIS, P.; CLAUSEN, T.; HUI, J.; GNAWALI, O.; KO, J. *RFC 6206*: *The Trickle Algorithm*. Internet Engineering Task Force (IETF), March 2011. Disponível em: http://ietf.org/rfc/rfc6206.txt. Acesso em 10 jan. 2018.

[RFC6347] RESCORLA, E.; MODADUGU, N. *RFC 6347*: *Datagram Transport Layer Security Version 1.2*. Internet Engineering Task Force (IETF), January 2012. Disponível em: http://ietf.org/rfc/rfc6206.txt. Acesso em 10 jan. 2018.

[RFC6550] WINTER, T.; THUBERT, P.; BRANDT, A.; HUI, J.; KELSEY, R.; LEVIS, P.; PISTER, K.; STRUIK, R.; VASSEUR, JP.; ALEXANDER, R. *RFC 6550*: *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. Internet Engineering Task Force (IETF), March 2012. Disponível em: http://www.ietf.org/rfc/rfc6550.txt. Acesso em 10 jan. 2018.

[RFC6551] VASSEUR, JP.; KIM, M.; PISTER, K.; DEJEAN, N.; Barthel, D. *RFC 6551*: *Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks*. Internet Engineering Task Force (IETF), March 2012. Disponível em: http://www.ietf.org/rfc/rfc6551.txt. Acesso em 10 jan. 2018.

[RFC6719] GNAWALI, O.; LEVIS, P. *RFC 6719*: *The Minimum Rank with Hysteresis Objective Function*. Internet Engineering Task Force (IETF), September 2012. Disponível em: http://www.ietf.org/rfc/rfc6719.txt. Acesso em 10 jan. 2018.

[RFC6552] THUBERT, P. *RFC 6552: Objective function zero for the routing protocol for low-power and lossy networks*. Internet Engineering Task Force (IETF), March 2012. Disponível em: http://www.ietf.org/rfc/rfc6552.txt. Acesso em 10 jan. 2018.

[RFC7159] BRAY, T. *RFC 7159*: *The JavaScript Object Notation (JSON) data interchange format*. Internet Engineering Task Force (IETF), March 2014. Disponível em: http://www.ietf.org/rfc/rfc7159.txt. Acesso em 10 jan. 2018.

[RFC7252] SHELBY, Z.; HARTKE, K.; BORMANN, C. *RFC 7252*: *The Constrained Application Protocol (CoAP)*. Internet Engineering Task Force (IETF), June 2014. Disponível em: http://www.ietf.org/rfc/rfc7252.txt. Acesso em 10 jan. 2018.

[RFC7641] HARTKE, K. *RFC 7641*: *Observing Resources in the Constrained Application Protocol (CoAP)*. Internet Engineering Task Force (IETF), September 2015. Disponível em: http://www.ietf.org/rfc/rfc7641.txt. Acesso em 10 jan. 2018.

[RFC7228] BORMANN, C.; ERSUE, M.; KERANEN, A. *RFC 7228*: *terminology for constrained-node networks*. Internet Engineering Task Force, May 2014. Disponível em: http://www.ietf.org/rfc/rfc7228.txt. Acesso em 10 jan. 2018.

[SED2011] SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4. ed. Addison-Wesley, 2011.

[SHE2010] SHELBY Z.; BORMANN C. *6LoWPAN*: The Wireless Embedded Internet, Wiley Series on Communications Networking & Distributed Systems. John Wiley & Sons, 2010.

[TI2012] TEXAS INSTRUMENTS INCORPORATED. *SWRS096D – CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications. 2012*. Disponível em http://www.ti.com/lit/ds/symlink/cc2538.pdf. Acesso em 10 jan. 2018.

[YAP2011] YAP, K.; YIAKOUMIS, Y.; KOBAYASHI, M.; KATTI, S.; PARULKAR, G.; Mckeown, N. *Separating authentication, access and accounting*: *A case study with OpenWiFi*. Open Networking Foundation, 2011.

[YI2013] YI, J.; CLAUSEN, T.; IGARASHI, Y. *Evaluation of routing protocol for low power and Lossy Networks: LOADng and RPL*. In: WIRELESS SENSOR (ICWISE), December 2013. IEEE, 2013, p. 19-24.

[ZHA2014] ZHANG, T.; LI, X. *Evaluating and Analyzing the Performance of RPL in Contiki*. In: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON MOBILE SENSING, COMPUTING AND COMMUNICATION, August 2014. ACM, 2014, p. 19-24.