INSTITUTO FEDERAL DE SANTA CATARINA

ANDRÉ FELIPPE WEBER

Roteamento para Estender o Tempo de Vida de Redes de Sensores sem Fios Utilizando Redes Definidas por Software

São José - SC

Dezembro/2018

ROTEAMENTO PARA ESTENDER O TEMPO DE VIDA DE REDES DE SENSORES SEM FIOS UTILIZANDO REDES DEFINIDAS POR SOFTWARE

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Tiago Semprebom, Dr. Eng.

Coorientador: Prof. Eraldo Silveira e Silva, Dr. Eng.

São José - SC

Dezembro/2018

André Felippe Weber
Roteamento para Estender o Tempo de Vida de Redes de Sensores sem Fios Utilizando Redes Definidas por Software/ André Felippe Weber. – São José - SC, Dezembro/2018-
71 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Tiago Semprebom, Dr. Eng.

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, Dezembro/2018.

1. Redes de Sensores Sem Fio. 2. QoS. 3. SDN. I. Prof. Tiago Semprebom, Dr. Eng.. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Roteamento para Estender o Tempo de Vida de Redes de Sensores sem Fios Utilizando Redes Definidas por Software

ANDRÉ FELIPPE WEBER

ROTEAMENTO PARA ESTENDER O TEMPO DE VIDA DE REDES DE SENSORES SEM FIOS UTILIZANDO REDES DEFINIDAS POR SOFTWARE

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 15 de outubro de 2015:

Prof. Tiago Semprebom, Dr. Eng., Dr.
Orientador
Instituto Federal de Santa Catarina

Prof. Eraldo Silveira e Silva, Dr. Eng.
Coorientador
Instituto Federal de Santa Catarina

Prof. Marcelo Maia Sobral, Dr. Eng.
Instituto Federal de Santa Catarina

Professor
Instituto Z

AGRADECIMENTOS

Agradeço à minha família que sempre incentivou e me concedeu a oportunidade de poder me dedicar em tempo integral aos estudos ao longo de todo o curso.

Agradeço também aos amigos que estiveram ao meu lado nos momentos de estudo e descontrações.

Por fim, agradeço ao IFSC e à todos os professores que contribuíram com a minha formação nos últimos anos, em especial aos meus orientadores cuja ajuda foi essencial para o sucesso deste trabalho.

epigrafe

RESUMO

As Redes de Sensores sem Fios (RSSFs) tornaram-se um recurso valioso no contexto da Internet das Coisas (IoT). Diversos domínios de aplicação como militar, hospitalar, de segurança e de sistemas de controle utilizam dados escalares e multimídia, produzidos por inúmeros módulos espalhados em uma região. As limitações de bateria, poder de processamento e memória, em conjunto com a heterogeneidade dos módulos utilizados, resultam em um complexo processo de gerenciamento da rede. Como consequência, atender aos requisitos de baixo consumo energético, tempo de vida e roteamento eficiente de dados nas RSSFs torna-se um desafio. O conceito de Redes Definidas por Software (SDN), possibilita minimizar a complexidade de gerenciamento dessas redes ao centralizar as decisões de roteamento e administração da RSSF em um controlador separado, fisicamente, da rede. Deste modo, este trabalho propõe implementar uma RSSF baseada no conceito de SDN, que utiliza a plataforma SDN-WISE para estender o tempo de vida da rede, através do uso diferentes algoritmos de roteamento e modos de operação dos módulos sensores.

Palavras-chave: Redes de Sensores Sem Fios, roteamento, IoT, SDN.

ABSTRACT

Wireless Sensor Networks (WSN) have become a valuable asset to the Internet Of Things (IoT). Application of such networks for military, health, security and control systems purposes may use thousands of sensor nodes to gather scalar and multimedia information about a specific region. The battery, processing and memory limitation along with heterogeneity of the deployed sensor nodes impose a complex network management. Thus, to meet the low energy consumption, lifetime and routing efficiency requirements in WSNs have become a challenging task. The use of Software Defined Networking (SDN) in WSNs reduces the complexity of the network management by centralizing the routing decisions on a remote controller. Therefore, it is proposed in this monograph the deployment of a WSN based on SDN that uses SDN-WISE to prolong the lifetime of the network through the use of diverse routing algorithms and sensor nodes operation modes.

Keywords: Wireless Sensor Network. routing. IoT, SDN.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SN Sensor Node 44

TCP *Transmission Control Protocol* 50

TD *Topology Discovery* 37

TTL *Time To Live* 35

UDGM *Unit Disk Graph Model* 61

SUMÁRIO

1 INTRODUÇÃO

O conceito de Internet das Coisas (IoT) tem sido amplamente investigado pela comunidade acadêmica nos últimos anos e atualmente ganha espaço nos setores de desenvolvimento das empresas de tecnologia. De acordo com o conceito de IoT, objetos comuns do cotidiano, como lâmpadas e sensores são conectados à Internet, permitindo, por exemplo, o controle de eletrodomésticos e a disponibilização de dados como umidade e temperatura de ambientes na Internet.

As aplicações de sensoriamento na IoT possuem o desafio de organizar uma rede com centenas de dispositivos. Por isso, as RSSF são um recurso valioso da IoT (CAPELLA et al., 2016). As RSSFs tendem a executar uma função colaborativa onde os sensores proveem dados, que são processados (ou consumidos) por módulos especiais chamados de sorvedouros (*sink nodes*). Estes módulos sensores geram dados escalares (pressão, temperatura, umidade, etc) e multimídia através de suas câmeras, microfones e sensores. Transmitem essas grandezas utilizando transceptores acoplados aos módulos, e são limitados em termos de memória, processamento e bateria.

Com a popularização da IoT, as dificuldades relacionadas à manutenção das aplicações e do roteamento das RSSF ficaram ainda mais latentes, principalmente, devido às limitações de *hardware* dos módulos. Dessa forma, existe a demanda no desenvolvimento de técnicas de gerenciamento que melhorem a escalabilidade das RSSFs. O conceito de Redes Definidas por Software (SDN) surge como uma das técnicas utilizadas nas RSSFs para gerenciamento da topologia, consumo energético e roteamento da rede para satisfazer às exigências como largura de banda e eficiência energética das aplicações (NDIAYE; HANCKE; ABU-MAHFOUZ, 2017).

O principio básico da SDN é a separação da rede em plano de controle e plano de dados. Onde o primeiro é responsável pelas decisões de roteamento e o segundo pelo encaminhamento dos dados. Dessa forma, um controlador centralizado, em um servidor remoto, controla e gerencia o encaminhamento realizado pelos módulos no plano de dados. Por exemplo, um controlador de uma RSSF cuja aplicação promove o monitoramento ambiental de uma região de difícil acesso e, que portanto exige a longevidade dos módulos, pode utilizar algoritmos complexos de roteamento para garantir, em tempo real, que os caminhos escolhidos contribuam com a manutenção do tempo de vida da rede.

O protocolo SDN-WISE se baseia na arquitetura SDN para reduzir a complexidade de configuração e gerenciamento das RSSFs. Desta forma, um controlador separado fisicamente do restante da rede executa algoritmos de roteamento complexos em um servidor remoto, enquanto módulos da rede encaminham os dados de um nó *source* até o *sink*, de acordo com a tabela de roteamento recebida do controlador.

A utilização da abordagem SDN economiza energia dos módulos e permite o uso de algoritmos complexos de roteamento que visam atender as necessidades impostas pelas aplicações da rede. Deste modo, este trabalho propõe um sistema que utiliza o protocolo SDN-WISE para capturar informações do estado dos enlaces da RSSF e enviar para um controlador remoto (na nuvem), que executa um algoritmo de roteamento. A Figura 1 ilustra uma RSSF que pode utilizar o sistema proposto. Cada módulo da RSSF envia, através do protocolo SDN-WISE, informações sobre seu nível de bateria, topologia da rede e estados dos enlaces para um controlador SDN, hospedado na nuvem. O controlador, por sua vez, executa um algoritmo de roteamento que tem como objetivo estender o tempo de vida da rede. Por fim, após a execução do algoritmo o controlador envia uma mensagem de atualização para os módulos da rede que podem alterar, remover ou adicionar novas rotas de encaminhamento de pacotes.

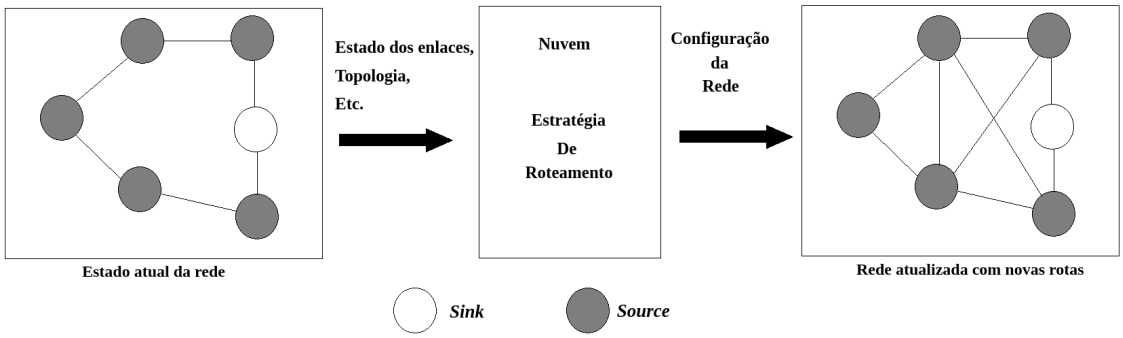


Figura 1 – Modelo do sistema.

1.1 Objetivos Gerais

O objetivo geral deste trabalho é implementar roteamento em uma rede de sensor sem fio, baseada no conceito de Redes Definidas por Software (SDN) com o propósito de estender o tempo de vida da rede.

1.2 Objetivos específicos

- Aplicação sobre a plataforma SDN-WISE no simulador Cooja;
- Conceber e implementar um controlador SDN que crie regras de roteamento com base em uma configuração estática e, nas informações recebidas da rede;
- Implementar algoritmo de roteamento por caminhos disjuntos.
- Implementar algoritmo de roteamento por caminhos encontrados através da aplicação de técnicas de recompensa negativa.
- Avaliação do uso de caminhos com custo mínimo, disjuntos e com recompensa negativa.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados temas essenciais para a compreensão e desenvolvimento deste trabalho. Na Seção 2.1 é explicado como as RSSFs são utilizadas para dar forma a muitas aplicações de sensoriamento da IoT. Na Seção 2.2 é apresentado o conceito de SDN. Na sequência, na Seção 2.3, o conceito de SDN é explorado no contexto de RSSFs, para tanto, a plataforma SDN-WISE será usada como referência. Finalmente, a aplicabilidade desta plataforma em uma aplicação será exemplificada na Seção 2.4.

2.1 Redes de Sensores para a Internet das Coisas

A IoT considera que os objetos comuns do cotidiano se comuniquem através da Internet. Esta ideia já pode ser observada na prática nos dispositivos inteligentes, como televisores, celulares, relógios, lâmpadas e sensores. Porém o aumento do número de dispositivos conectados à Internet acarreta no crescimento do volume de dados gerados e consequentemente impõe o desafio de rotear, de maneira eficaz, os dados gerados na rede.

O desafio de organizar uma rede com centenas de dispositivos é uma realidade, principalmente quando se trata de aplicações de sensoriamento. Por isso, as RSSFs são um recurso valioso da IoT (CAPELLA et al., 2016). Criadas na década de 1950, as RSSFs foram no princípio utilizadas pelo exército norte americano para detectar a movimentação de submarinos soviéticos (NASIR; KU-MAHAMUD, 2006). Com o avanço tecnológico elas passaram a ser implementadas em zonas terrestres e subterrâneas, além das subaquáticas. Dentre as aplicações mais citadas em publicações atuais estão aquelas que abrangem a área da saúde, de controle de tráfego de veículos, monitoramento e militar.

As RSSF atuais são compostas por diversos módulos equipados com processador, memórias *Read Only Memory* (ROM) e *Random Access Memory* (RAM), rádio transceptor, um ou mais sensores e uma fonte de energia. Algumas redes utilizam estações base com melhor poder de processamento e capacidade energética para receber e tratar os dados gerados pelos módulos. Dessa forma, é necessário garantir que os módulos comuniquem-se entre si e com a estação base, mesmo que indiretamente, a fim de realizar a entrega de pacotes a qualquer destinatário da rede e garantir que as exigências de QoS sejam cumpridas (GUY, 2006).

O conjunto de métricas de performance utilizadas para definir a qualidade do serviço nas RSSFs variam de acordo com a necessidade de cada aplicação. Porém, segundo Nasir e Ku-Mahamud (2006), existem oito métricas que são comumente utilizadas: (I) vazão, (II) atraso fim-a-fim, (III) taxa de sucesso na entrega de pacotes, (IV) perda de pacotes, (V) distância medida através da quantidade de saltos (*hops*) entre a fonte e o destinatário, (VI) consumo e eficiência energética, (VII) carga da rede e (VIII) tempo de vida.

A topologia de rede comumente utilizada para organizar as RSSFs é a Ponto-a-Ponto. Como mostrado na Figura 2, a comunicação das redes ponto-a-ponto é normalmente realizada entre os vizinhos mais próximos. Os integrantes da rede podem atuar como fonte, receptor ou roteador dos dados já que para realizar a comunicação com um destino fora de alcance é necessário rotear os dados através dos vizinhos. Devido às limitações de processamento e de consumo de energia dos módulos a comunicação entre vizinhos é a principal razão para o uso desta topologia nas RSSFs.

Atualmente, verifica-se que parte da atenção destinada ao estudo das RSSFs é dirigida ao

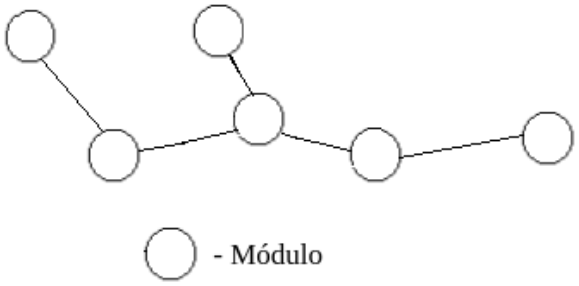


Figura 2 – Topologia ponto-a-ponto.

estudo de redes de sensores com capacidade de processar dados multimídia. Isto se deve ao avanço tecnológico e à redução do custos de aquisição dos módulos e outros componentes utilizados nas Rede de Sensores Multimídia sem Fio (RSMSF)s. As RSMSFs são compostas por módulos equipados com câmeras, microfones e outros sensores produtores de conteúdo multimídia (SHAKSHUKI; XING; MALIK, 2009), além de módulos equipados somente com transceptores utilizados no roteamento de dados na rede.

As RSMSFs atraíram atenção dos pesquisadores da área devido ao seu potencial científico e aplicações atrativas (SHEN; BAI, 2016). Por exemplo, a distribuição de câmeras ao longo de uma rodovia assegura a redundância do sistema de videomonitoramento, disponibiliza múltiplos pontos de vista e, através da escolha de câmeras mais próxima ou com qualidade superior, amplia o ângulo de visão de um evento. A adição de microfones ao sistema permite detectar e determinar a localização aproximada de sons incomuns ao ambiente como barulhos de tiros (ANG et al., 2013) ou freadas bruscas.

Devido ao maior fluxo de dados gerado pelas aplicações implementadas nas RSMSFs em comparação com as RSSFs as métricas de QoS que atuam diretamente na garantia de entrega e integridade dos dados multimídia ganham maior relevância em detrimento do consumo de energia que, nas redes RSSFs, ocupa o posto de destaque (HAMID; HUSSAIN, 2014; AKYILDIZ; MELODIA; CHOWDURY, 2007). As exigências de QoS nas RSMSFs podem variar de uma aplicação para outra. Por exemplo, uma aplicação de controle de fluxo de tráfego é mais tolerante à falhas na entrega de pacotes que uma aplicação militar. Porém, de forma geral, as métricas de QoS utilizadas nas redes RSMSFs visam a garantia de largura de banda, tempo de vida da rede, controle do *jitter*[1], taxas de atraso toleráveis, controle de erro através de técnicas de codificação de fonte de baixa complexidade e estratégias de melhoria da eficiência energética (HAMID; HUSSAIN, 2014).

2.1.1 Tecnologias para Redes de Sensores sem Fios

2.1.1.1 IEEE 802.15.4

Dentre as tecnologias desenvolvidas e implementadas nas RSSFs está o padrão IEEE 802.15.4 que especifica as camadas física (PHY) e de Controle de Acesso ao Meio (MAC) para redes *Low-Rate Wireless Personal Area Network* (LR-WPAN) e, portanto se caracteriza pela comunicação sem fio com baixa taxa de transmissão de dados, baixo consumo energético e custo de implementação. Segundo Semprebom (2012), apesar de não ter sido projetada especificamente para as RSSFs, o IEEE 802.15.4 vem sendo amplamente adotado e possibilitou o surgimento de novos produtos baseados nesta tecnologia.

Na implementação de uma RSSF utilizando a tecnologia IEEE 802.15.4 os módulos podem ser categorizados, de acordo com sua capacidade de processamento ou função na rede, em dois tipos:

[1] Variação no atraso

- **Dispositivo de Função Completa (FFD)**: Obrigatório em uma LR-WPAN, este dispositivo suporta até três modos de operação:

  1. **Coordenador PAN**: O dispositivo fica responsável por controlar a *Personal Area Network* (PAN) e identificar a rede para que outros dispositivos se associem (SILVA, 2017).
  2. **Coordenador**: Dispositivo que deve, obrigatoriamente, estar associado à um Coordenador PAN e oferece o serviço de sincronização através da transmissão de *beacons*.
  3. **Dispositivo simples**: Dispositivo que atua apenas como um nó na rede e, portanto não implementa as funções de coordenador PAN ou coordenador.

- **Dispositivo de Função Reduzida (RFD)**: Dispositivo associado a um único FFD e que opera com implementação miníma do IEEE 802.15.4.

No padrão IEEE 802.15.4 é possível utilizar as topologias estrela e ponto-a-ponto. Na topologia estrela existe um único nó central ao qual todos os módulos se conectam, como mostrado na Figura 3. Este dispositivo central deve ser um FFD operando como Coordenador PAN que fica responsável por iniciar uma nova rede com um identificador único e rotear todos os pacotes sendo transmitidos. Os outros integrantes são RFDs ou FFDs operando como Coordenador ou Dispositivo simples (SEMPREBOM, 2012).

O nó central da topologia estrela pode ser visto como um ponto crítico de falha, uma vez que toda a comunicação da rede é realizada através dele. Por isso, devido às limitações energéticas dos FFDs, o padrão IEEE 802.15.4 recomenda que o Coordenador PAN esteja conectado em alguma fonte de alimentação, e que esta topologia seja usada em aplicações que apresentem facilidade na substituição da fonte de energia do Coordenador PAN, como automação residencial e industrial (SILVA, 2017).

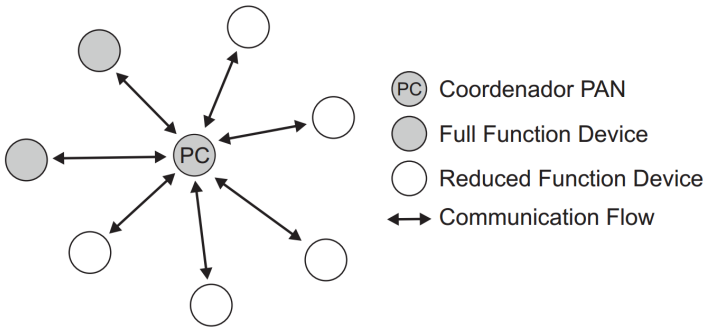


Figura 3 – Topologia estrela (SEMPREBOM, 2012)

A topologia ponto-a-ponto (Figura 4), assim como a estrela, possui um único coordenador PAN. Porém é mais confiável já que, por ser descentralizada, possui múltiplos caminhos para o roteamento dos pacotes. Por não possuir um nó central, os módulos se comunicam diretamente com os vizinhos em seu raio de cobertura, e através de múltiplos saltos com os vizinhos mais distantes.

As funcionalidades de roteamento por múltiplos saltos não são definidas pelo padrão IEEE 802.15.4, pois devem ser definidas na camada de rede (SEMPREBOM, 2012). As funções de roteamento utilizadas para realizar os múltiplos saltos possuem a desvantagem de introduzir complexidade e reduzir a longevidade da rede, uma vez que os módulos precisam reencaminhar dados recebidos dos vizinhos. Porém, tem como vantagem o aumento do alcance da rede.

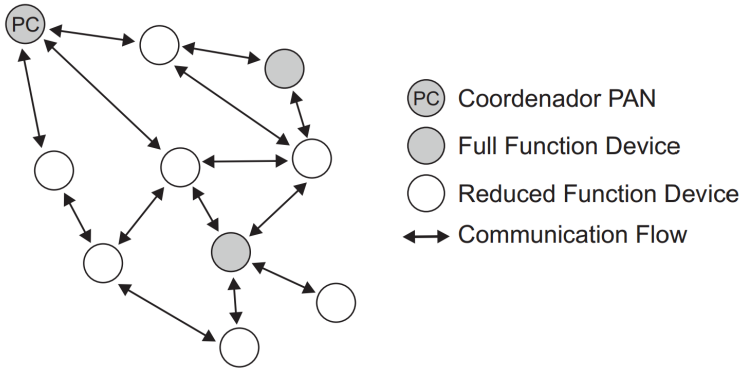


Figura 4 – Topologia ponto-a-ponto (SEMPREBOM, 2012)

2.1.1.2 *Long Range* (LoRa)

Outra tecnologia que se destaca na criação de RSSFs é o LoRa. Esta é uma tecnologia de camada física e uma modulação para comunicação sem fio que possibilita enlaces de longa distância, baixo consumo energético e taxa de transmissão. Segundo Saraiva (2017), quanto menor a taxa de transmissão, maior a duração da mensagem (tempo do sinal no ar) e, consequentemente maior a energia de bit. Como resultado, há o aumento da qualidade de recepção do sinal (cerca de -150 dBm), proporcionando uma maior cobertura, estimada na ordem de 10 a 15 quilômetros em áreas rurais e 2 a 5 quilômetros em áreas urbanas. A frequência de operação do LoRa varia por região porém, assim como o IEEE 802.15.4 operam na faixa Industrial, Cientifico e Médico (ISM), mais especificamente em 2.4 GHz, 868/915 MHz, 433 MHz e 169 MHz. A taxa de transmissão está na ordem de centenas de bits ou dezenas de kilobits por segundo (CENTENARO et al., 2016).

O LoRaWAN define o protocolo de comunicação e a arquitetura do sistema para a rede, enquanto a camada física LoRa possibilita o uso de enlaces de longa distância. Apesar do longo alcance, as redes ponto-a-ponto apresentam a desvantagem do consumo desnecessário de bateria no módulos que atuam como roteadores da rede. Por isso, no LoRaWAN adotou-se a topologia estrela de longa distância. O que permite uma maior economia energética para conexões de longa distância.

As redes LoRaWAN são implementadas com o objetivo de otimizar redes *Low Power Wide Area Network* (LPWAN) para aplicações IoT. As LPWANs oferecem excelente eficiência energética com expectativa de vida útil maior que dez anos para as baterias, e são pensadas para aplicações que transmitem pequenas quantidades de dados, por grandes distâncias e poucas vezes por hora (ALLIANCE, 2015).

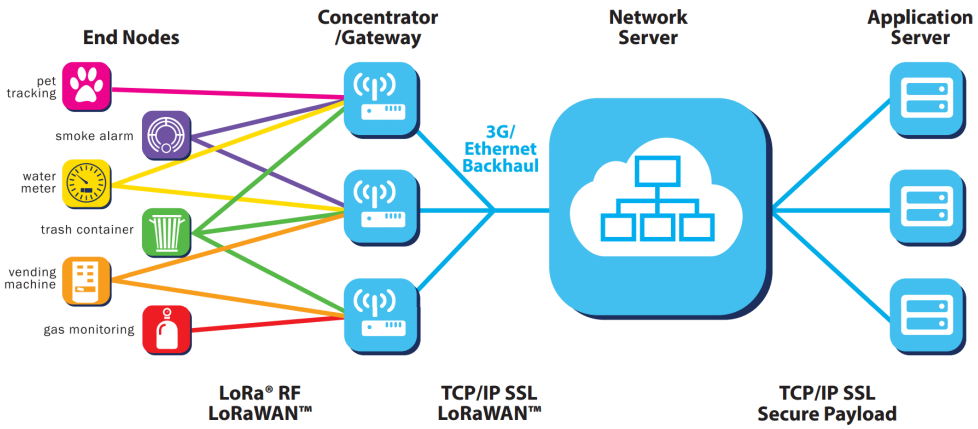


Figura 5 – Visão geral de uma rede LoRaWAN com topologia em estrela (ALLIANCE, 2015)

De acordo com Alliance (2015), as redes LoRaWAN são compostas por *gateways* que atuam como nó central da rede e módulos (*end-devices*) que compõem a rede, como mostrado na Figura 5. São classificados de acordo com a relação entre disponibilidade de recepção de dados e vida útil de sua bateria:

- **Classe A:** São sensores, energizados via bateria, com a melhor eficiência energética da rede LoRaWAN. Os *end-devices* de classe A transmitem somente quando necessário e recebem dados apenas após realizar uma transmissão. Dessa forma, ao realizar uma transmissão o *end-device* espera por duas janelas curtas de tempo e volta a desabilitar seu rádio transceptor.
- **Classe B:** São atuadores, energizados via bateria, que além de implementar as janelas de recepção do *end-devices* classe A também agendam janelas de tempo para recepção de dados. Para isso é necessário a sincronização entre *gateway* e *end-device* classe B que, é realizada através de *beacons* enviados pelo *gateway*.
- **Classe C:** São atuadores conectados a uma fonte externa de energia que, aceitam recepção o tempo todo com exceção para quando há uma transmissão a ser realizada. Esse *end-device* tem a vantagem de eliminar latência para recepção de dados, porém necessita de uma fonte contínua de energia.

A topologia em estrela é a ideal para as rede LoRaWAN por causa do longo alcance proporcionado pela camada física dos dispositivos LoRa (OLIVEIRA, 2017). Esta topologia permite uma melhor eficiência energética quando comparada à topologia ponto-a-ponto, adotada nas redes de curto alcance como as redes IEEE 802.15.4, visto que nenhum módulo atua como roteador. E para contornar o ponto de falha único, característico da topologia em estrela, usa-se redundância de *gateways* (ALLIANCE, 2015).

Diferentemente do que ocorre na topologia estrela tradicional, em que há apenas um nó central em que todos os módulos se conectam, na LoRaWAN existem diversos nós implementados como *gateways*. Assim sendo, segundo Alliance (2015) as mensagens produzidas por um módulo não são recebidas por apenas um, mas sim por múltiplos *gateways*. Cada *gateway* encaminha os dados gerados pelo *end-device* até um servidor responsável por tratar dos dados, eliminar dados redundantes, administrar a taxa de dados, etc. Dessa forma, através da redundância criada por múltiplos *gateways* elimina-se o ponto crítico da rede, que é dispor de um único nó central.

Em Alliance (2015) atribuiu-se às redes LoRaWAN a característica de gerenciar a longevidade da bateria, capacidade e segurança da rede, onde:

- **Longevidade da bateria:** Os módulos na LoRaWAN são assíncronos, ou seja, não há troca de mensagem entre *end-devices* ou com o *gateway* para manter sincronia. Logo, ganha-se em eficiência energética, já que o processo de sincronização é o fator número um de redução da vida das baterias (ALLIANCE, 2015).
- **Capacidade da rede:** A capacidade da rede LoRaWAN depende, principalmente, da sua característica de adaptação da taxa de dados (*Adaptive Data Rate* (ADR)) que permite que um *gateway* receba mensagens de um grande volume de *end-devices* (ALLIANCE, 2015). Isto ocorre pois o uso da *Chirp Spread Spectrum* (CSS) na camada física permite que um *gateway* receba dados com diferentes taxas de transmissão no mesmo canal e, simultaneamente. Assim, é possível que dispositivos mais próximos do *gateway* operem com uma taxa de transmissão mais alta e com potência menor que dispositivos mais distantes (SARAIVA, 2017).

  O ADR auxilia também no aumento da capacidade da rede já que ele possibilita que seja aumentado a taxa de dados, de forma a reduzir o tempo de transmissão de um módulo e dar oportunidade para

que outros transmitam. Além do mais, é possível reduzir o consumo energético de um módulo ao reduzir sua potência e taxa de dados.

- **Segurança da rede:** Na LoRaWAN é utilizado criptografia *Advanced Encryption Standard* (AES) e garante-se segurança nas camadas de aplicação e rede. Na camada de aplicação os dados do usuário final são protegidos enquanto na de rede garante-se a autenticidade do módulo (ALLIANCE, 2015).

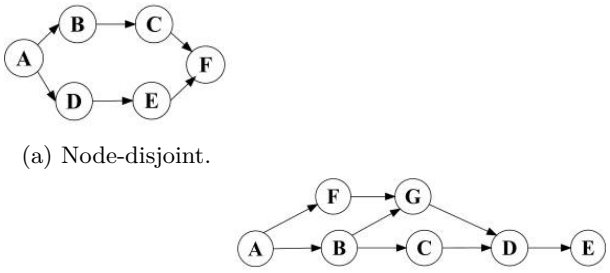
2.1.2 Técnicas de Roteamento em RSSFs

Devido às limitações de recursos dos módulos utilizados nas RSSFs é importante projetar protocolos de roteamento que possibilitem a transmissão de pacotes por longas distâncias na rede. Para isso, alguns protocolos utilizam técnicas de múltiplos caminhos e saltos para criar e manter rotas de encaminhamento dos pacotes. Como resultado, o protocolo melhora a eficácia da comunicação entre módulos e em alguns casos, com a Internet.

Muitos protocolos de roteamento para RSSFs foram propostos nas últimas décadas considerando as limitações energéticas, de processamento e memória dos módulos, assim como as exigências das aplicações e arquitetura das redes de sensores sem fio (ALAZZAWI; ELKATEEB, 2008). Em muitas aplicações de RSSFs espera-se que a rede tenha capacidade de organização automática, uma vez que os módulos são distribuídos aleatoriamente e se comportam como componentes de uma rede *ad hoc*. Porém, o roteamento dos dados nas RSSFs não pode ser feito como em redes sem fio tradicionais. Segundo Singh, Singh e Singh (2010) as RSSFs se diferenciam das redes celulares e da Rede *Ad hoc* móvel (MANET) por possuírem algumas características especiais:

- Grandes quantidades de dispositivos na rede. Algumas redes podem ser significantemente mais povoadas que as redes MANET;
- Limitação energética e computacional dos módulos contrastante com o que acontece nas redes sem fio tradicionais;
- Auto-organização dos módulos implementada por algumas tecnologias utilizadas nas RSSFs, como o Zigbee, que utiliza o padrão IEEE 802.15.4 nas camadas inferiores (PHY e MAC) e implementa as camadas superiores introduzindo a capacidade de auto configuração e auto-recuperação de falhas;
- Os módulos não são confiáveis, uma vez que podem apresentar falha devido às condições ambientais em que são utilizados e por esgotamento da bateria;
- Possível redundância na transmissão de dados correlatos devido à proximidade entre módulos equipados com sensores;
- Uma RSSF é normalmente criada para uma aplicação específica, considerando que as métricas de QoS e exigências da rede variam entre as aplicações;
- A topologia da rede muda constantemente devido à falta de confiabilidade dos módulos, interferências e possível movimentação dos módulos.

Tradicionalmente, protocolos de roteamento utilizam a abordagem *single-path* para escolha de caminhos. Ou seja, baseado em uma métrica de performance ou na menor distância, um único caminho é escolhido. Porém, através da abordagem de roteamento por múltiplos caminhos é possível melhorar a resiliência da rede, ou seja, a capacidade que a rede possui de se recuperar de falhas, visto que protocolos que utilizam a abordagem *single-path* interrompem o roteamento por algum tempo ou por completo em um evento de falha. Segundo Al-karaki e Kamal (2004), o preço que se paga ao melhorar a resiliência



(a) Node-disjoint.

(b) Link-disjoint.

(c) Partially-disjoint.

Figura 6 – Categorias de protocolos de múltiplos caminhos, de acordo com o grau de separação dos caminhos. (RADI et al., 2012)

da rede é o aumento de tráfego e consumo energético na rede, visto que mensagens periódicas serão transmitidas para manter os caminhos ativos e atualizados.

Diferentes abordagens podem ser utilizadas para projetar um algoritmo que faz uso de roteamento por múltiplos caminhos. Por exemplo, pode-se priorizar a eficiência energética ou a confiabilidade da rede. Assim, é possível optar por evitar rotear pacotes por módulos que estão com baixo nível de bateria, dividir um pacote por múltiplos caminhos ou enviar o mesmo pacote por diversos caminhos. Porém, todos os protocolos de roteamento por múltiplos caminhos compartilham três etapas básicas: Descoberta, seleção e manutenção dos caminhos (RADI et al., 2012).

Quanto à descoberta dos caminhos, os protocolos podem ser divididos em três categorias de acordo com o grau de separação dos caminhos (*disjointedness*).

- ***Node-disjoint*:** Os caminhos escolhidos não possuem nenhum módulo em comum, além do fonte e sorvedouro. Ou seja, são completamente independentes um dos outros. Esse tipo de caminhos possuem a desvantagem de apresentar pontos de falha em caso de perda de comunicação em um dos enlaces entre módulos. Por exemplo, na Figura 6a, é possível observar que em caso de falha do módulo C, módulo B também não consegue mais transmitir seus pacotes para o sorvedouro F.

- ***Link-disjoint*:** Os caminhos possuem módulos em comum, porém não compartilham os enlaces, como mostrados na Figura 6b.

- ***partially-disjoint*:** Os caminhos compartilham seus módulos e enlaces, como mostrado na Figura 6c . Possui a principal desvantagem de ser vulnerável às falhas desses componentes da rede. Porém, é possível criar diversos caminhos redundantes que melhoram a resiliência da rede (RADI et al., 2012).

A seleção dos caminhos ocorre de acordo com a necessidade da aplicação. No caso das RSSFs é interessante que os caminhos sejam escolhidos levando em conta o estado da bateria dos módulos que encaminham os pacotes. Porém, pode-se optar por melhorar a confiabilidade da rede. Para isso, os pacotes podem ser transmitidos de forma redundante por múltiplos caminhos. Outra possibilidade seria transmitir os dados por apenas um dos caminhos, deixando o restante dos caminhos como *backup* para eventos de falha de módulos ou enlaces.

A manutenção dos caminhos pode ser feita, segundo Radi et al. (2012), em um dos seguintes três eventos: (1) Quando um dos caminhos falha, (2) Quando todos os caminhos falham ou (3) Quando um certo número de caminhos falham. Alguns protocolos classificados como *on-demand* permitem que a manutenção da rede seja realizada em tempo real, através da descoberta e o constante monitoramento dos

caminhos. Nessa classificação, existem os *proactive on-demand* que descobrem e mantêm os caminhos ao longo da vida útil da rede, e os *reactive on-demand* que descobrem um novo caminho somente quando necessário, isto é, antes de uma transmissão de dados (Mbarushimana e Shahrabi (2007)).

2.2 *Redes Definidas por Software (SDN)*

As infraestruturas de redes tradicionais são compostas por dispositivos como roteadores e *switches* que executam algoritmos complexos e, muitas vezes com interfaces de controle limitadas e específicas do fabricante do equipamento. Consequentemente, a manutenção e atualização dos protocolos da rede (e.g. IPV6), torna-se um desafio (NUNES et al., 2014). O SDN foi projetado para facilitar o gerenciamento das redes de forma flexível e programática. Isto é feito ao retirar a complexidade de controle da rede dos equipamentos (*hardware*) e repassá-la para uma aplicação controladora, em *software* (KLAUBERG, 2016).

O principio básico do SDN é a separação da rede em planos de controle e dados (Figura 7). Onde o primeiro é responsável pelas decisões de encaminhamento e o segundo pelo encaminhamento dos dados:

- ***Camada de controle*:** Esta camada é composta por um controlador que possui uma visão geral da rede (Camada de Dados) e capacidade para centralizar toda a inteligência da rede, isso é, gerenciar o roteamento, topologia, segurança, QoS e controlar o consumo energético da rede (NDIAYE; HANCKE; ABU-MAHFOUZ, 2017). Desta forma, os módulos da camada inferior enviam requisições, através de uma *application programming interface* (API) para o controlador sempre que uma nova rota é necessária. O controlador, que pode estar em um servidor remoto responde com a informação necessária para completar a tabela de roteamento do módulo.
- ***Camada de dados*:** Os equipamentos (*hardware*) que executam uma mesma ou diferentes aplicações são atribuídos à camada de dados. Estes módulos são responsáveis por realizar o sensoriamento da rede e rotear dados de acordo as informações recebidas pelo controlador.

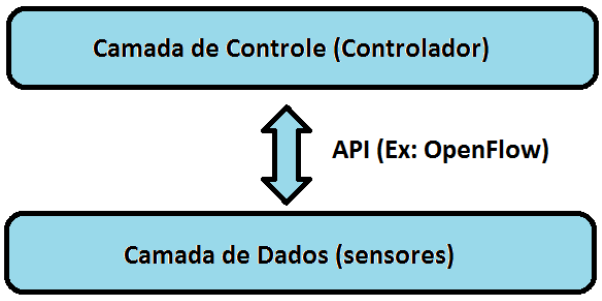


Figura 7 – Arquitetura do *Software Defined Networking*

O *OpenFlow* é a principal API utilizada nas redes SDN para padronizar a comunicação entre as camadas de controle e dados. Os dispositivos da camada de dados chamados de *OpenFlow switch* possuem uma ou mais tabelas de roteamento (*flow table*) e um canal de comunicação segura com o controlador (KLAUBERG, 2016).

As tabelas de roteamentos são compostas por regras de correspondência, estatísticas e ações. Quando um pacote é recebido, as regras de correspondência são comparadas com informações presentes no cabeçalho, porta de ingresso ou metadado do pacote. Se alguma das regras é validada, então a ação correspondente da *flow table* é disparada. No caso de não haver correspondências, uma regra conhecida como *table-miss* é utilizada para definir o destino do pacote, que pode ser descartado, encaminhado para

outra *flow table* ou transmitido para o controlador. Por fim, as informações de estatística incluem dados como número de pacotes e *bytes* recebidos (NUNES et al., 2014).

O controlador da rede utiliza o canal de comunicação segura, através do protocolo *OpenFlow*, para criar, remover ou modificar regras da *flow table*. O que pode ser feito de maneira reativa (em resposta ao pacote recebido) ou proativa. Por sua vez, os *OpenFlow switches* utilizam o canal no caminho inverso para requisitar instruções de processamento de um pacote que não possui um regra de correspondência.

O *Forwarding and Control Element Separation* (ForCES) é outra API que assim como o *OpenFlow* prevê a criação das camadas de controle e dados e padroniza a comunicação entre elas. Criado pela *Internet Engineering Task Force* (IETF), o padrão instancia uma entidade (*Forwarding Element* (FE)) para a camada de dados e outra (*Control Element* (CE)) para a de controle. No ForCES o FE possui um bloco (*Logical Function Block* (LFB)) que o controlador utiliza para configurar o *hardware* e definir como os pacotes serão processados (NUNES et al., 2014). Porém, diferentemente do que ocorre no *OpenFlow* as camadas de dados e controle continuam coexistindo no mesmo *hardware*. Por esse motivo, esta proposta é menos utilizado na criação de redes SDN, se comparado ao *OpenFlow*.

2.3 SDN em redes de sensores: SDN-WISE

Apesar de atenderem às necessidades de comunicação dos módulos, as RSSFs se tornam complexas de gerenciar e configurar a medida que o número de componentes aumenta. Dessa forma, a utilização da arquitetura do SDN apresenta-se como uma alternativa interessante devido à possibilidade de retirar a complexidade do processo de roteamento dos módulos limitados em bateria, processamento e memória e entregá-la para a camada de controle. Além disso, segundo Ndiaye, Hancke e Abu-Mahfouz (2017), o potencial visto no SDN se deve à simplificação do gerenciamento, além da reconfiguração da rede devido à existência de um controlador centralizado.

Aplicações de sensoriamento ambiental, por exemplo, que detectam queimadas, poluição e temperatura podem exigir o uso de centenas ou até milhares de sensores espalhados em ambientes críticos, sem acesso à fontes de energia e com instabilidade climática. Portanto, a viabilidade da aplicação depende da facilidade de configuração, roteamento e manutenção dos dispositivos. Por isso, é evidente que o uso de um controlador centralizado, que possui visão completa da topologia da rede e capacidade de tomada de decisões para garantir o funcionamento da rede facilita e viabiliza a criação de amplas redes de sensores sem fio.

2.3.1 Visão Geral

O SDN se popularizou graças ao *OpenFlow* que consolidou-se como um dos primeiros protocolos SDN à definir uma API de comunicação entre Camadas de Controle e Dados. Porém, o *OpenFlow* foi pensado para redes cabeadas e, portanto, não pode ser utilizado para amenizar as dificuldades encontradas no gerenciamento das RSSFs em virtude das limitações de *hardware* (memória, bateria e processamento), e da heterogeneidade de dispositivos.

Assim sendo, em Galluccio et al. (2015), os autores se basearam no *OpenFlow* para criar o SDN-WISE, que define uma arquitetura SDN para RSSFs e portanto, permite criar e comunicar controladores da rede, módulos sorvedouros (*Sink*) e fontes (*Source*). O SDN-WISE, diferentemente do *OpenFlow*, implementa funcionalidades que melhoram a eficiência energética da rede. Por exemplo, é possível desligar o rádio dos módulos periodicamente (*duty-cycle*), permite-se que os módulos agreguem suas mensagens à pacotes oriundos dos vizinhos, que são roteados através deles. Além do mais, o protocolo SDN-WISE prevê que os módulos possuam capacidade de decisão em alguns casos. Dessa forma, diminui-se a quantidade de

interações entre os módulos e o controlador melhorando a eficiência energética da rede.

O comportamento esperado dos módulos que compõe uma rede que utiliza o SDN-WISE foi descrito em Galluccio et al. (2015). Os módulos possuem três estruturas de dados: (i) *WISE States Array*, (ii) *Accepted IDs Array* e (iii) *WISE Flow Table*, utilizadas para armazenar as informações de roteamento recebidas do controlador da rede.

O SDN-WISE foi projetado para operar em modo *stateful*. Dessa forma, os módulos são projetados como uma máquina de estados para reagir a um determinado evento de acordo com o estado em que ele se encontra. A vantagem desta abordagem é a redução da quantidade de interações com o controlador considerando que o módulo ganha capacidade de reagir a um dado evento com base em informações locais. Por conseguinte, o *WISE States Array* é utilizado para armazenar os estados de cada módulo da rede.

Todos os pacotes transmitidos na rede devem possuir uma identificação (ID) que é utilizada nos módulos receptores para verificar se o pacote deve ser processado ou descartado por eles. Esta verificação é possível pois existe uma lista de IDs, chamada de *Accepted IDs Array*, que são comparados ao ID recebido no pacote. Se o ID recebido combina com um dos IDs armazenados, então o receptor processa o pacote de acordo com a instrução repassada pelo controlador. Caso contrário, o pacote é descartado.

Caso haja a necessidade de processar o pacote, o receptor recorre à *WISE Flow Table* para encontrar uma regra que indique uma ação correspondente, que deve ser tomada para realizar o processamento. Se nenhuma regra for encontrada, então o receptor deve requisitar uma nova regra ao controlador. Portanto, o roteamento no SDN-WISE ocorre de acordo com regras de correspondência definidas na *WISE Flow Table*. Quando as condições definidas nas regras de correspondência são atendidas, uma ação correspondente é executada e as estatísticas em relação à regra são atualizadas.

Matching Rule					Matching Rule					Matching Rule					Action					Statistics	
Op.	Size	S	Addr.	Value	Op.	Size	S	Addr.	Value	Op.	Size	S	Addr.	Value	Type	M	S	Addr.	Value	TTL	Counter
=	2	0	2	B	>	2	0	10	x_{Thr}	=	1	1	0	0	Modify	1	1	0	1	122	23
=	2	0	2	B	≤	2	0	10	x_{Thr}	=	1	1	0	1	Modify	1	1	0	0	122	120
=	2	0	2	B	-	0	-	-	-	-	0	-	-	-	Forward	0	0	0	D	122	143
=	2	0	2	A	=	1	1	0	0	-	0	-	-	-	Drop	0	0	-	-	100	42
=	2	0	2	A	=	1	1	0	1	-	0	-	-	-	Forward	0	0	0	D	100	32

Figura 8 – *WiseFlowTable* (GALLUCCIO et al., 2015)

Dessa forma, são definidas três seções na *WISE Flow Table*, como ilustrado na Figura 8. A primeira seção é a de Regras de Correspondência (*Matching Rules*), composta por até três condições (colunas de *Matching Rule*), que se atendidas, disparam uma ação. O campo S (*State*) da Regra de Correspondência especifica se a regra é aplicável ao pacote atual ou ao estado do módulo. Por exemplo, um módulo A que recebe um pacote de B pode executar a ação de acordo com uma informação contida neste pacote ou com base no estado em que ele se encontra devido à um dado recebido anteriormente de um terceiro módulo C. Os campos Addr. (*Address*) e *Size* indicam respectivamente: O primeiro *byte* e o tamanho da cadeia de caracteres do pacote ou do estado que devem ser consideradas na regra. O campo Op. (*Operator*) define o operador relacional a ser utilizado em conjunto com o campo *Value* para verificar se a regra é valida (GALLUCCIO et al., 2015).

A segunda seção da *WISE Flow Table* define a ação a ser executada quando as condições da seção Regra de Correspondência são atendidas. O campo *Type* da seção *Action* pode assumir o valor:

- *Forward to*: Para encaminhar o pacote;

- *Drop*: Para descartar o pacote;
- *Modify*: Para modificar um valor do pacote;
- *Send to INPP*: Para encaminhar o pacote para a camada *In-Network Packet Processing* (INPP);
- *Turn off*: Para desligar o rádio do módulo.

Os campos *Address* e *Size* auxiliam a execução da ação, isto é, complementam a ação indicando de acordo com o tipo o endereço do módulo para o próximo salto, probabilidade de descarte, onde modificar o pacote ou o estado além o tipo de processamento a ser realizado na INPP e o tempo restante até que o rádio seja desligado. O campo S, assim como na seção Regra de Correspondência, indica se a ação é referente ao pacote ou ao estado. Por fim, o campo M indica se ação é exclusiva ou não. Ou seja, se outra ação deve ser executada após a atual ou se o processamento deve ser finalizado.

A terceira seção armazena as estatísticas referentes ao uso das regras de correspondência, que são utilizadas para verificar a validade das regras. O campo *Time To Live* (TTL) armazena o tempo restante da validade da regra e o campo *Counter* indica a quantidade de vezes que a regra foi utilizada.

2.3.2 Estrutura dos Pacotes

Os pacotes (*WISE Packets*) projetados para uso nas RSSFs com SDN-WISE possuem no mínimo dez *bytes* de comprimento. Os campos do cabeçalho (Figura 9) são organizados da seguinte maneira:

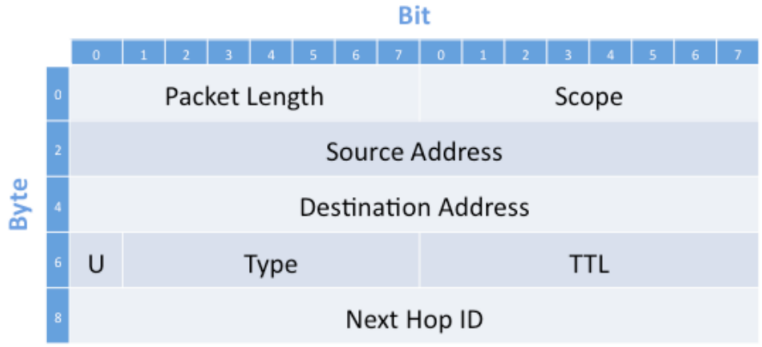


Figura 9 – Cabeçalho do *Wise Packet* (GALLUCCIO et al., 2015).

- ***Packet length*:** Indica o tamanho total do pacote (incluindo o *payload*);
- ***Scope*:** Indica um ou mais controladores da rede que têm interesse no pacote;
- ***Source Address* e *Destination Address*:** Indicam o endereço de origem e destino, respectivamente;
- ***Flag U*:** Indica que o pacote deve ser entregue para o sorvedouro mais próximo;
- ***Type of packet*:** Indica o tipo do pacote e determina como ele será processado pelo receptor. Existem oito tipos de pacotes. Dessa forma, além de dados o pacote pode carregar informações da topologia da rede, configuração entre módulos e controlador, configuração da *WISE Flow Table* e instruções para desligamento do rádio por um determinado intervalo de tempo;
- ***TTL*:** Indica o tempo de vida do pacote sendo decrementado de um a cada salto;
- ***Next Hop ID*:** É o ID utilizado pelo receptor para verificar no *Accepted IDs Array* se o pacote deve ou não ser processado por ele.

Todos os pacotes transmitidos internamente na rede ou para o controlador possuem o mesmo formato do cabeçalho, porém informações distintas no *payload*. A diferença entre eles é destacado no campo *Type* que como mostrado em SDN-WISE (2017b) pode assumir um dos seguintes valores:

0 ***Data*:** Pacote composto apenas pelo cabeçalho e *payload*;

1 ***Beacon*:** Transmitido através de um *broadcast*, o pacote é utilizado para que os módulos compartilhem informações de bateria e distância do sorvedouro;

2 ***Report*:** Utilizado para manter o controlador atualizado sobre o estado dos enlaces na rede. O pacote, transmitido pelos módulos da rede, é composto por informações da vizinhança (endereço e indicador *Received signal strength indication* (RSSI)), além da sua distância do sorvedouro e nível de bateria.

3 ***Request*:** Pacote utilizado para transmitir ao controlador uma requisição de regra de correspondência. Ou seja, a transmissão deste pacote é realizada quando não há uma regra para tratar um pacote recém recebido pelo módulo. Para isso, o módulo encapsula o pacote que não possui regra de correspondência no *payload* da *Request*, e o envia para o controlador.

4 ***Response*:** Pacote, transmitido pelo controlador, que contém a resposta à requisição de uma nova regra de correspondência.

5 ***OpenPath* :** Este tipo de pacote tem como finalidade diminuir a quantidade de pacotes transmitidos na rede. Pois ao invés de enviar uma regra de roteamento para cada módulo, o controlador cria um caminho de roteamento e transmite, neste pacote, os endereços de todos os módulos que compõem o caminho. Ao receber o pacote, os módulos criam regras de correspondências com base nos endereços dos módulos que compõem o caminho.

6 ***Config*:** Utilizado na configuração da rede.

7 ***RegProxy*:** Pacote transmitido do sorvedouro para o controlador, para notificar a sua existência.

2.3.3 Camada de Dados

Na camada de dados do SDN-WISE os módulos são classificados como sorvedouros (*Sink*) ou nodo (*Node*), que pode gerar dados (fonte) ou apenas encaminhá-los na rede. A diferença na arquitetura dos dois (Figura 10) é que o sorvedouro possui um módulo extra (ADAPT.) que é utilizado para adaptar os pacotes da rede para um protocolo externo afim de possibilitar a comunicação com o controlador da rede. Além do ADAPT. outras três camadas foram projetadas para os módulos da camada de dados:

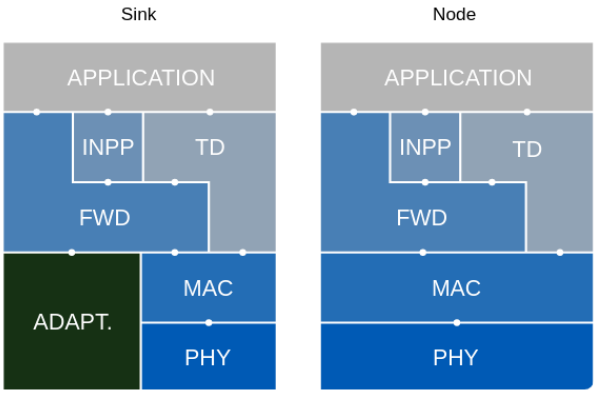


Figura 10 – Arquitetura dos módulos sorvedouro (*sink*) e fonte (*source*) (SDN-WISE, 2017a).

- ***In-Network Packet Processing*** **:** O INPP contribui para a eficiência energética do módulo, pois é nesta camada que é realizada a agregação de dados. O módulo pode agregar sua própria mensagem à pacotes que são roteados através dele e que serão entregues a um destino em comum;

- ***Topology Discovery*** **(TD):** A camada TD possui um protocolo que coleta informações geradas pelos módulos sobre a topologia da rede e as disponibiliza para o controlador. Adicionalmente, esta camada é responsável por descobrir e manter atualizado o próximo salto do roteamento de cada módulo em direção a um dos sorvedouros da rede. Assim, os sorvedouros são responsáveis por enviar periodicamente *Beacons* que atuam como pacotes de descoberta de topologia (*TD packet*), através de *broadcast*, para os nodos da RSSF. Neste pacote são enviados o ID do sorvedouro, informação de bateria e um contador inicialmente zerado que indica a distância em saltos que um módulo está do sorvedouro.

  De acordo com Galluccio et al. (2015), o protocolo utilizado no TD pode ser descrito em 4 etapas. Supondo que um módulo A recebe um *TD packet* de um módulo B (que pode, ou não, ser sorvedouro):

  1. O módulo A cria, se já não existe, uma referência para B como um de seus vizinhos, armazenando as informações de RSSI e nível de bateria de B.
  2. O módulo A verifica se já não recebeu um *TD packet* que contenha informação de distância menor. Caso a informação de distância recebida no último pacote seja menor, então uma nova rota mais eficiente foi encontrada. Caso contrário, o módulo A ignora a informação de distância e pula as próximas duas etapas.
  3. O módulo A atualiza a informação de bateria do pacote com seu nível de bateria.
  4. O módulo A retransmite o pacote recebido de B.

  Os nodos transmitem, periodicamente, informações sobre a sua vizinhança para o controlador da rede. A taxa de envio dessas informações deve ser escolhida de forma a encontrar um equilíbrio entre manter o controlador atualizado e evitar uma grande quantidade de pacotes na rede (*overhead*).

- **Forwarding (FWD):** O protocolo da camada FWD é o responsável por receber os pacotes e realizar o processamento deles de acordo com as informações recebidas no *header* do pacote e com as disponíveis na *WISE flow table*.

2.4 Gerenciamento de uma RSSFs com SDN-WISE

Uma das diferenças a ser destacada no SDN-WISE em relação ao *OpenFlow* é a sua capacidade de armazenar estados. Como já mencionado, reagir à recepção de um pacote de acordo com o estado do receptor diminui o número necessário de comunicações entre módulo e controlador, o que afeta positivamente a eficiência energética da rede. Ademais, como demonstrado em Dio et al. (2016) os estados podem ser utilizados para representar o congestionamento percebido pelo módulo.

A ideia proposta em Dio et al. (2016) é utilizar o tamanho da fila do *buffer* de recepção, no qual os pacotes são enfileirados até que sejam processados, para definir os estados do módulo. Por exemplo, um módulo que esteja com sua fila vazia, deve manter-se em um estado inicial *Green* (G). Já quando a fila está cheia, o estado deve ser modificado para *Red* (R) e o roteamento deve ser alterado para evitar perda de pacotes. Para isso, o controlador cria regras de roteamento com base no congestionamento percebido pelo módulo e na categorização dos fluxos da rede em k classes com prioridades distintas, onde C_1 possui a mais baixa prioridade e C_K a mais alta. Desta maneira, é possível diferenciar a maneira como os fluxos são tratados oferecendo garantias de entrega dos pacotes transmitidos.

O mecanismo apresentado propõe o descarte de pacotes e balanceamento do tráfego da rede. No primeiro caso, é utilizada uma probabilidade menor de descarte dos pacotes oriundos de fluxos com prioridade alta, em detrimento daqueles com menor prioridade, de forma a garantir mais recursos da rede para os fluxos mais criticos, ou seja, com maior prioridade. No segundo caso o controlador deve sugerir caminhos alternativos ou múltiplos caminhos para evitar o ponto de congestionamento da rede.

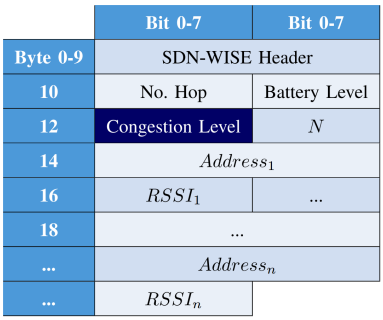


Figura 11 – novo pacote SDN-WISE report (DIO et al., 2016).

Para integrar essa solução de QoS ao SDN-WISE é proposto em Dio et al. (2016), que um campo *Congestion Level* seja adicionado ao pacote *Report*, como mostrado na Figura 11. E que, três estados, cujas transições ocorrem de acordo com o tamanho da fila no *buffer* de recepção sejam utilizados para determinar como um pacote deve ser processado:

1. ***Green* (G):** Representa o estado inicial da fila, em que ela está vazia. Ou seja, não há congestionamento na rede e portanto não há descarte de pacotes ou necessidade de caminhos alternativos. A transição para o próximo estado ocorrerá somente quando a fila alcançar o limite definido por T_{GY}.

2. ***Yellow* (Y):** Representa o estado em que há pacotes em espera na fila. Porém, o tamanho da fila ainda não atingiu o limite máximo, definido por T_{YR}. Neste estado, o módulo receptor passa a descartar pacotes com uma certa probabilidade, repassada pelo controlador, de acordo com a prioridade do fluxo. Isso é feito com objetivo de garantir a entrega dos pacotes oriundos de fluxos com alta prioridade.

3. ***Red* (R):** Representa o estado em que a fila está cheia. Neste caso, a probabilidade de descarte dos pacotes oriundos de fluxos de baixa prioridade é ainda maior que o repassado pelo controlador para o estado Y.

Um possível cenário de uso para a abordagem proposta em Dio et al. (2016) é mostrado na Figura 12. Onde em um primeiro momento T1 apenas os módulos do tipo A transmitem seus pacotes para o sorvedouro. Estes pacotes têm prioridade C_3 e são roteados através do módulo 1, que está no estado G. Em um segundo momento T2, o módulo do tipo B inicia sua transmissão utilizando um fluxo com prioridade C_1, o que faz com que o limite T_{GY} do módulo 1 seja atingido e o estado seja alterado para Y.

Como mostrado na Figura 13, a partir desse momento a probabilidade de descarte dos pacotes transmitidos dos módulos A passa de 0% para 5%. Os pacotes do módulo B que têm a mais alta prioridade são descartados com probabilidade de 1%. Em um terceiro momento T3, o módulo C inicia sua transmissão com prioridade C_2, o que faz com que o módulo 1 fique completamente congestionado e, transite seu estado para R. Com isso, as probabilidades de descarte aumentam para todas as classes de fluxos, sendo que na *Option 2*, o descarte dos pacotes pode chegar a 80% para os fluxos com baixa prioridade. Porém, mesmo neste cenário mais crítico os fluxos com alta prioridade possuem garantia de tráfego.

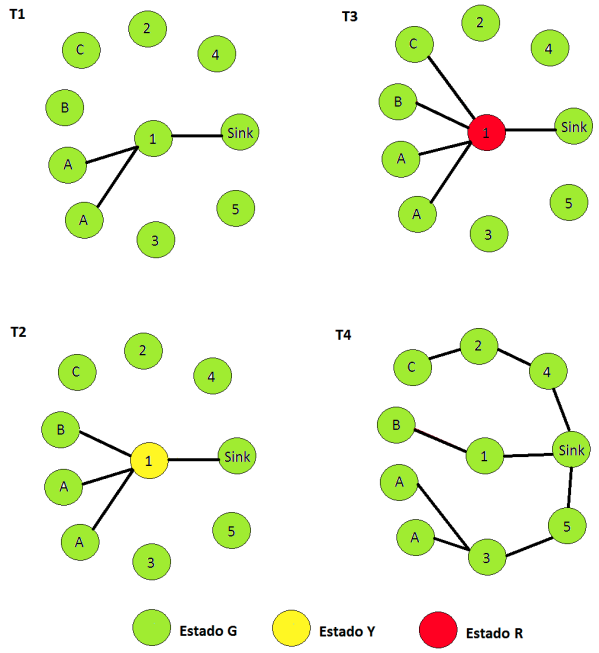


Figura 12 – Possível cenário de congestionamento e garantia de QoS utilizando SDN-WISE.

A transmissão periódica dos estados dos módulos para o controlador permite que o roteamento seja realizado dinamicamente. Dessa forma, diante do cenário apresentado em T3, o controlador pode, por exemplo, definir que os fluxos com prioridade C_3 sejam desviados para um caminho alternativo. O mesmo pode ser feito para os fluxos com prioridade C_2. Resultando na reserva do melhor caminho, escolhido originalmente para o fluxo do módulo B, que possui prioridade C_1, como mostrado em T4, na Figura 12.

			Option 1	Option 2
	Green State	**Yellow State**	**Red State**	**Red State**
C_1	NO DROP	1%	5%	10%
C_2	NO DROP	3%	20%	45%
C_3	NO DROP	5%	40%	80%

Figura 13 – Probabilidades de descarte de pacotes de acordo com a classe do fluxo. (DIO et al., 2016)

3 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos relacionados onde RSSFs são criadas e sobre elas aplica-se o conceito de SDN. Com isso, os autores almejam configurar a rede e criar regras de roteamento a partir de um controlador remoto. No final do capítulo os trabalhos descritos serão comparados entre si.

3.1 Conservação de energia em RSSFs utilizando SDN

No trabalho realizado por Smitha e Annapurna (2017) as camadas de controle e dados são claramente separadas com o intuito de conservar energia dos módulos sensores. Segundo os autores, a separação da rede em camadas como ocorre na integração do SDN à RSSF resulta em economia de energia. Isso ocorre pois o consumo dos nodos é reduzido em comparação com as RSSFs tradicionais, onde os módulos sensores são responsáveis por encaminhar dados e encontrar caminhos de roteamento, enquanto no SDN apenas pelo encaminhamento.

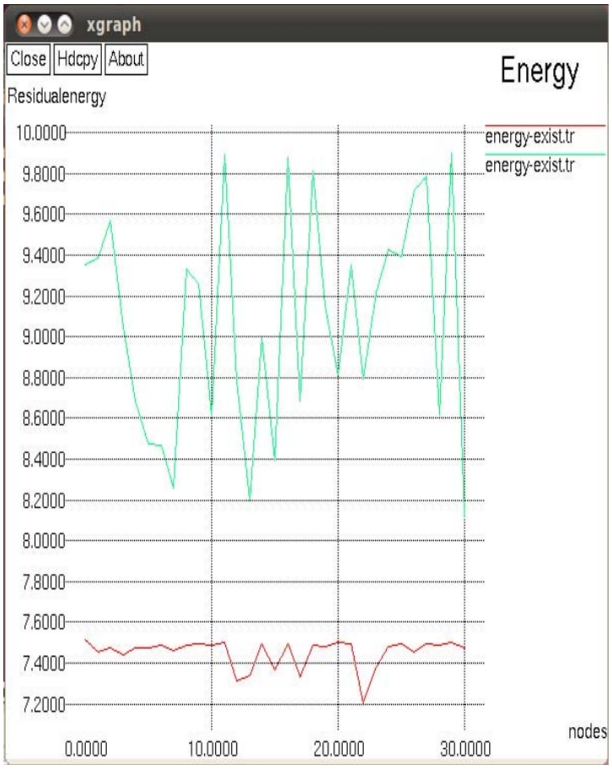


Figura 14 – Energia residual dos módulos integrantes de uma RSSF tradicional (vermelho), e de uma RSSF utilizando SDN (Azul) (SMITHA; ANNAPURNA, 2017).

Na implementação da RSSF utilizando SDN a comunicação entre as camadas de controle e dados foi realizada através do *OpenFlow*. O controlador substitui o sorvedouro da rede, enquanto o restante dos módulos são organizados em *clusters*. O controlador escolherá um líder (*cluster head*) para cada *cluster*, considerando a capacidade energética do módulo e a distância média dele em relação ao restante dos módulos. Os módulos do *cluster* deverão destinar os dados sensoreados ao *cluster head*. Desta forma, os participantes de cada *cluster*, exceto pelo *cluster head*, ficam responsáveis apenas por encaminhar dados com base na sua tabela de roteamento (*flow table*).

Após o recebimento de informações do nível energético e localização de cada módulo da rede o controlador gera uma única tabela de roteamento e a destina ao *cluster head*, que a repassa ao restante dos módulos. Por exemplo, Na *flow table* mostrada na Figura 15 há regras que serão utilizadas apenas pelo *cluster head* (CH) e outras destinadas para os demais nodos do *cluster* além de conter ações correspondentes e estatísticas sobre os uso das regras.

Como resultado, os autores demonstram, através de um gráfico (Figura 14) de módulos (eixo X) por energia residual (eixo Y), que a integração do SDN a uma RSSF mantém os níveis de energia residual dos módulos sensores acima dos níveis esperados para uma RSSF tradicional.

RULE	ACTION	STATISTICS
SOURCE = CH	DROP	D1
SOURCE != CH	FORWARD	F1
FWD NODE != CH	DROP	D2
FWD NODE = CH	FORWARD	F2
DEST = CONTROLLER	END	E1

Figura 15 – Tabela de roteamento gerada pelo controlador. (SMITHA; ANNAPURNA, 2017)

3.2 Mapa energético de RSSFs utilizando SDN

No trabalho realizado por Núnez e Margi (2017) os autores propõem a criação de um mapa energético para RSSFs baseadas no conceito de SDN. Para isso, o controlador implementa um modelo de previsão do consumo energético dos módulos da rede. Seguindo a arquitetura SDN, um controlador centralizado obtém informações do comportamento da rede e estima a taxa de consumo energético de cada módulo. Esta taxa é então utilizada para criar e atualizar periodicamente, um mapa energético da rede. Como resultado, os autores pretendem antecipar falhas na rede devido ao esgotamento energético dos módulos.

Na modelagem do mapa energético foram considerados quatro modos possíveis de operação para os módulos: Transmitindo, Escutando, Processando e Economia de Energia. Em seguida, cada modo é considerado como um estado de uma cadeia de Markov. Assim sendo, para uma cadeia de Markov com M estados, a probabilidade de transição de um estado i para um estado j, após n transições é definida pela equação de *Chapman-Kolmogorov* (Figura 16). Sendo que $X_n = i$ representa um módulo em modo de operação i e P_{ij} a probabilidade de transição do estado i para j.

$$P_{ij}^{n} = \sum_{k=1}^{M} P_{ik}^{r} P_{kj}^{(n-r)} \qquad for \;\; 0 < r < n$$

Figura 16 – Equação 1 - *Chapman-Kolmogorov*. (NÚNEZ; MARGI, 2017)

Para o estado atual $i(X_0 = i)$, o total de períodos t que um módulo permanecerá em um estado s, durante T períodos, é definido por (Figura 17):

Se um módulo está no estado i e E_s é a energia dissipada durante um período de tempo no estado s, então a previsão de energia gasta pelo módulo nos próximos T períodos é expressa por (Figura 18):

$$\sum_{t=1}^{T} P_{is}^{t}$$

Figura 17 – Equação 2 (NÚNEZ; MARGI, 2017)

$$E^{T} = \sum_{s=1}^{M} (\sum_{t=1}^{T} P_{is}^{t}) E_{s}$$

Figura 18 – Equação 3 (NÚNEZ; MARGI, 2017)

A matriz (MxM), que representa a probabilidade de transição entre os quatro estados do modelo é montada por cada módulo de acordo com suas transições de estados e o período que um estado é mantido. A matriz é então enviada, juntamente com o nível energético do módulo para o controlador que monta um mapa energético da rede. Com essa informação, o controlador utiliza um algoritmo para prever a taxa de consumo energético de cada módulo, com isso, os módulos apenas atualizam o controlador caso haja algum consumo anormal de energia, para que então o controlador atualize seu algoritmo de previsão de consumo energético.

3.3 *Sleep Scheduling* em RSSFs utilizando SDN

No trabalho realizado por Wang et al. (2016) é proposto um algoritmo intitulado SDN-ECCKN, que tem como objetivo reduzir o tempo total das transmissões realizadas durante a vida útil de uma RSSF. Isso é feito através de um controlador SDN que determina períodos de atividade (*awake*) e inatividade (*sleep*) dos módulos da rede. Segundo Wang et al. (2016) o princípio de um mecanismo do tipo *Sleep Scheduling* é automaticamente e deliberadamente, reduzir o consumo energético de certos módulos, ou seja, reduzir o número de mensagens trocadas, enquanto mantém outros módulos em plena atividade em um dado intervalo de tempo. Com isso, os autores afirmam que seu algoritmo pode melhorar o tempo de vida da rede, aumentar a quantidade de módulos vivos e diminuir a quantidade de módulos isolados e portanto sem uso na rede quando comparado ao algoritmo *Energy Consumed uniformly-Connected K-Neighborhood* (EC-CKN).

No algoritmo EC-CKN o qual é tomado como base para o desenvolvimento do SDN-ECCKN, cada módulo deve possuir até K módulos vizinhos ativos, que também possuam K módulos vizinhos ativos. Se um módulo verificar que essa condição é falsa, então ele permanecerá ativo, caso contrário entrará em modo *sleep*. Para isso, cada módulo deve enviar para sua vizinhança o seu nível de energia residual (Eu), a quantidade de vizinhos que ele possui e, o nível residual energético de cada um destes vizinhos. Ao receber essas informações, um módulo deve verificar, antes de mudar seu estado para *sleep*, se ele possui K vizinhos ativos que possuam energia residual maior que a sua própria (Eu) e ainda que esses vizinhos também possuam K outros vizinhos com energia residual maior que Eu (YUAN et al., 2011).

No algoritmo proposto (SDN-ECCKN), as transmissões *broadcast* feitas entre os módulos para que suas informações e de seus vizinhos sejam compartilhadas são eliminadas. Isso ocorre pois os módulos passam a transmitir um *beacon* com informações do seu nível de energia residual diretamente para o controlador SDN, ao invés de realizar dois *broadcasts* para seus vizinhos (com sua energia e com a energia de seus vizinhos). Adicionalmente, no EC-CKN cada módulo deve receber transmissões dos seus vizinhos, enquanto no SDN-ECCKN apenas os módulos que mudarão suas rotas ou seu estado receberão transmissões do controlador.

Ao fim dos experimentos o autores concluem que seu algoritmo é capaz de prolongar o tempo de vida da rede, além de estabilizar a média do consumo energético dos módulos, ou seja, o consumo de bateria se mantém mais linear que no EC-CKN. A quantidade de módulos vivos a cada intervalo de tempo em que há interação com o controlador também aumenta e finalmente a rede é melhor aproveitada, uma vez que menos módulos ficam isolados e portanto sem uso.

3.4 Estendendo o tempo de vida de uma RSSF mista

No trabalho realizado em Tomović e Radusinović (2016) os autores analisam o fato de as SDNs serem uma tecnologia emergente que não substituirá, repentinamente as RSSFs tradicionais. Por isso, eles consideram uma implementação mista em que módulos chamados de (Software Defined Networking Sensor Node (SDNSN)), habilitados a integrar uma RSSF baseada no conceito de SDN, coexistem com módulos (Sensor Node (SN)) que integram RSSFs tradicionais. Como resultado, os autores pretendem verificar se é possível melhorar o controle de tráfego e estender a tempo de vida de uma RSSF, considerando que nem todos os módulos podem ser controlados pelo controlador SDN.

***Algorithm 1**: Route calculation*

```
1:# G- reduced network graph used for route calculation
2:# src- source node, BS- base station, nodes- live nodes
3:# Costs - link costs, R - residual energy of source node
4:# EC - energy used to transmit and recevice data on the link
5: function: find_route(src, BS, nodes)
6:   for node in nodes do:
7:    if node.type is SDN do:
8:      for nb in node.neigbours do:
9:        G.addLink(node, nb)
10:       Costs([node, nb]) = EC([node, nb])^α / R([node])^β
11:     end for
12    else do:
13:     nb=node.MTE_nextHop
14:     G.addLink(node, nb)
15:     Costs([node,nb])=EC([node,nb])^α/R([node])^β
16:   end if
17: end for
18: return leastCostPath(G, src, BS, Costs)
19: end function
```

Figura 19 – Algoritmo. (TOMOVIĆ; RADUSINOVIĆ, 2016)

Como o controlador SDN não pode enviar rotas para os SNs, o algoritmo Minimum Transmission Energy (MTE) é utilizado. Este algoritmo escolhe o próximo salto para roteamento com base na distância do vizinho reduzindo o desperdício de energia ao encontrar o módulo mais próximo. Porém, ele exige conhecimento de todos os módulos na área de cobertura e por esta razão todos os módulos devem implementá-lo. Com isso, a rede montada para simulação é composta por módulos sensores tradicionais SN que executam o MTE, módulos sensores SDNSN que podem receber rotas do controlador SDN, mas que também executam o MTE, além de um controlador centralizado que envia caminhos de roteamento para os SDNSNs. O controlador necessita ter acesso a topologia completa da rede e por isso assume-se que ele possui conhecimento sobre a capacidade e o consumo energético, além da localização dos SNs e com isso pode estimar a energia residual destes módulos. Por fim, assume-se que o sorvedouro possui uma tabela especial que armazena estatísticas quanto as transmissões realizadas através dos SNs.

Na linha 7 do algoritmo mostrado na Figura 19 é possível observar que primeiramente os módulos são separados em SNs e SDNSNs. O custo do enlace é considerado como uma função da energia residual dos módulos (com peso α) e energia necessária para transmissão e recepção dos pacotes no enlace (com peso β). Por fim, o algoritmo de *Dijkstra* é utilizado para encontrar o caminho com menor peso. Assim sendo, quando o algoritmo (Figura 19) é aplicado à uma rede que possui apenas 20% de módulos SDNSNs, o tempo de vida de rede aumenta 49,35% em relação ao cenário em que apenas SNs são utilizados. Além do mais, os mesmos 20% de módulos SDNSNs utilizados aumentam em 21,86% o número de pacotes entregues com sucesso ao sorvedouro da rede. Por fim, observou-se que o tempo de vida aumenta proporcionalmente ao aumento do número de SDNSNs na rede.

3.5 Comparação

No trabalho descrito em 3.1, os autores utilizam o *OpenFlow*, sem nenhuma indicação de que foram feitas alterações para ajustá-lo à realidade das RSSFs, isto é, às limitações de *hardware* existentes. O que não é indicado, uma vez que o *OpenFlow* foi criado para redes cabeadas. Como pode ser observado na Tabela 1, o *overhead* introduzido na rede é mínimo se comparado aos demais trabalhos apresentados nesta seção, pois a tabela de roteamento é enviada pelo controlador durante a inicialização da rede e é mantida estática ao longo do funcionamento dela. Porém, o não monitoramento e o fato de não ser reativo às possíveis falhas devido ao esgotamento energético ou mobilidades dos módulos, por exemplo, torna esta abordagem menos confiável que os demais trabalhos, já que um dos módulos pode falhar sem que o controlador seja notificado. Os autores deste trabalho não divulgaram qual algoritmo é utilizado no controlador.

Trabalho	Monitoramento	Reativo	Preditivo	Overhead
3.1	Não Não Configurável	Não	Não	Não considerável Não Configurável
3.2	Não. Não Configurável.	Não	Sim	Não considerável Não Configurável
3.3	Sim Não é configurável	Sim	Não	Considerável Não Configurável
3.4	Sim, de forma parcial Não é configurável	Sim	Sim, de forma parcial	Considerável Não Configurável

Tabela 1 – Comparação entre os trabalhos.

Em 3.2 os autores conseguem aliar confiabilidade e baixo *overhead* na rede. Porém, também não há um monitoramento frequente da rede por parte do controlador, que apenas recebe atualizações se houver alguma variação inesperada no consumo energético de um dos módulos. Por isso, a confiabilidade é válida apenas para cenários em que as falhas ocorrem devido ao esgotamento energético dos módulos, o que exclui falhas devido à mobilidade ou intempéries, por exemplo. Os autores deste trabalho também não divulgaram qual algoritmo foi utilizado no controlador.

No trabalho 3.3 os autores propõem diminuir o consumo energético da rede ao introduzir um mecanismo de *Sleep Scheduling* na rede. Com isso, o controlador fica responsável por enviar caminhos de roteamento e comandos para que alguns módulos mudem seus estados para *sleep* e, desliguem seus rádios. Como consequência, há um aumento na quantidade de pacotes na rede, o que impacta diretamente na energia consumida. Porém, isto é contornado com roteamento e a diminuição considerável do consumo energético dos módulos quando estão em *sleep*, considerando que o rádio é a unidade que mais gasta energia nos módulos sensores.

Na proposta introduzida em 3.4 os autores demonstraram as vantagens da implementação de um controlador SDN em uma RSSF tradicional. Utilizando o algoritmo de Djikstra em uma rede em que os

módulos compatíveis com o SDN enviam periodicamente informações da topologia para o controlador e o *sink* se responsabiliza por gerar estatísticas sobre os módulos tradicionais, os autores conseguiram economizar 49,35% de energia em relação à uma RSSF tradicional, apesar do *overhead* causado pelo controlador SDN.

4 PROPOSTA

As restrições de *Hardware* dos nodos de uma RSSF limitam o desempenho dos algoritmos de roteamento executados sobre eles. O baixo poder de processamento impede que algoritmos complexos e que portanto produzem melhores rotas sejam executados e a baixa capacidade energética requer o mínimo uso possível dos componentes do nodo. Por isso, é proposto um sistema que utiliza o SDN para executar algoritmos de roteamento que visam estender o tempo de vida de uma RSSF.

A visão geral do sistema proposto, vista na Figura 20, mostra a criação de duas camadas (controle e dados), como previsto pelo SDN, que são ligadas através da *API* fornecida pelo SDN-WISE. Portanto, na camada de dados os módulos implementam o protocolo do SDN-WISE e, podem ser utilizados para gerar dados e roteá-los para o controlador, através de um sorvedouro da rede. Para isso, eles transmitem informações como nível energético e estado dos enlaces para o controlador e, em troca recebem caminhos de roteamento.

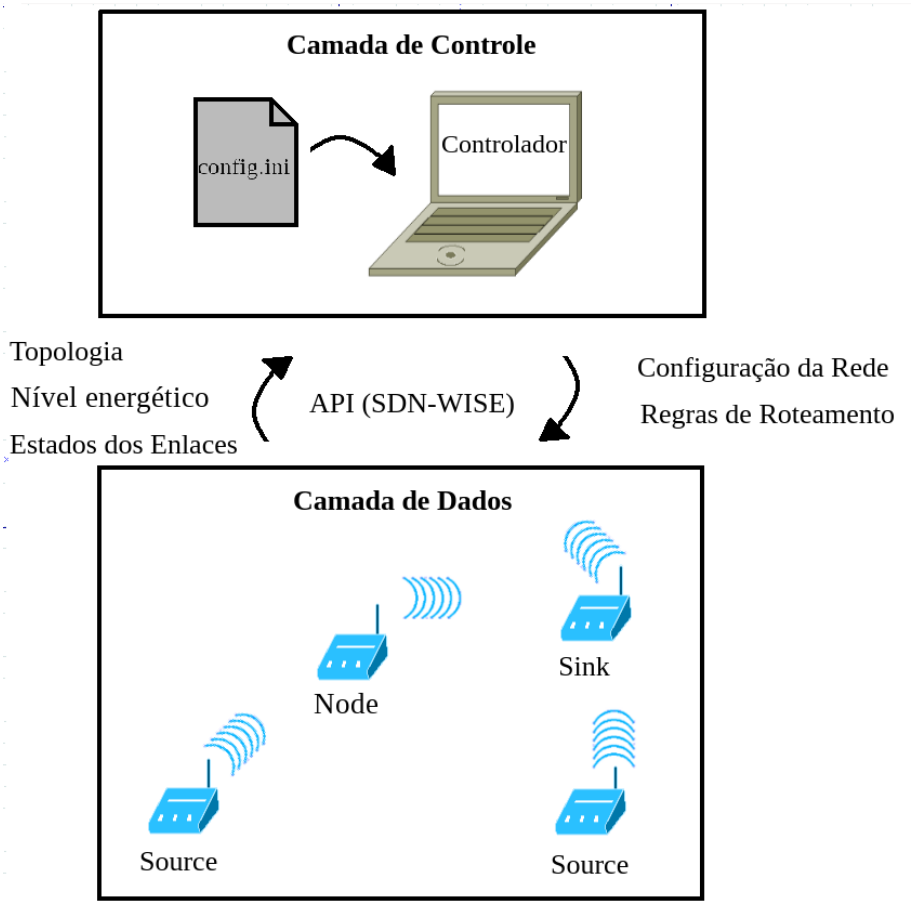


Figura 20 – Visão geral do sistema proposto.

Na camada de controle há um arquivo de configuração *config.ini* que é utilizado para definir parâmetros estáticos aplicáveis aos nodos da rede, ao cenário de simulação e ao próprio controlador que pode utilizá-los nos algoritmos de roteamento ou para definir o seu comportamento. Além das configurações estáticas, o controlador baseia-se nas informações recebidas sobre a topologia da rede para criar regras de roteamento. Dessa forma, a cada *Report* recebido o controlador atualiza um grafo em que as vértices representam os nodos e as arestas representam enlaces unidirecionais e o repassa para os algoritmos de

roteamento juntamente com os parâmetros estáticos.

Com isso, são propostos dois algoritmos que utilizam abordagens distintas com o mesmo propósito de estender o tempo de vida da rede. O que é essencial para reduzir os custos de manutenção e melhorar a eficiência energética dos módulos. A primeira abordagem utiliza o número de saltos entre os nodos fonte e sorvedouro como métrica, para encontrar caminhos disjuntos, em que nenhum nodo é compartilhado entre os diferentes caminhos de roteamento da rede, semelhante com o que ocorre na abordagem *Node-disjoint*, descrita na Subseção 2.1.2, porém sem o uso de múltiplos caminhos. Já a segunda abordagem utiliza o nível residual energético dos nodos como métrica para encontrar os caminhos, além de aplicar recompensas negativas (*Negative Reward*) aos enlaces em uso pela rede.

O intuito de criar caminhos disjuntos é garantir que o fluxo de dados sobre os nodos que compõem um caminho será sempre menor ou igual à capacidade de um dos nodos fontes de gerar dados. O que contribui com o aumento do tempo de vida dos nodos do caminho, já que o consumo de energia é limitado à carga gerada em um único fluxo de dados. Além do mais, o algoritmo proposto mantém o caminho ativo, se necessário, até que o nível energético de um dos módulos que o compõem ultrapasse um limiar mínimo energético aceitável. Com isso, o restante dos nodos da rede que estão ociosos podem entrar em modo de economia de energia.

A segunda abordagem de roteamento prevê o balanceamento do consumo energético da rede ao penalizar os enlaces que são utilizados em um caminho ativo, ou seja, o uso de nodos que já estejam roteando dados é desencorajado ao aplicar recompensas negativas, que neste caso, é dobrar o custo de uso dos enlaces do nodo em questão. Além do mais, são realizadas verificações e manutenções periódicas da rede para garantir que o caminho em uso sempre será aquele em que os módulos possuem melhor nível de energia residual. Dessa forma, esta abordagem contribui com a extensão da vida da rede já que os pacotes roteados são melhor distribuídos, em relação à primeira abordagem, entre os diversos nodos que compõem a RSSF.

5 IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo serão descritos as ferramentas utilizadas no desenvolvimento do trabalho. Dessa forma, a ferramenta de simulação de redes de sensores sem fio Cooja bem como o funcionamento do protocolo SDN-WISE nos módulos sensores e no controlador serão apresentados na Seção 5.1. Os pacotes pensados neste trabalho para auxiliar na configuração e eficiência energética da rede são descritos na Seção 5.2. A implementação dos nodos e controlador utilizados para avaliar os algoritmos propostos são descritos nas Seções 5.3 e 5.4, respectivamente. Os algoritmos de roteamento propostos são descritos da Seção 5.5 e então avaliados e comparados entre si e com o algoritmo de Djikstra implementado no controlador original do SDN-WISE na Seção 5.6.

5.1 Ferramentas Utilizadas

5.1.1 Simulador Cooja

O Cooja é um simulador de redes de sensores sem fio capaz de executar programas escritos para o sistema operacional Contiki a nível de sistema operacional (executando diretamente no CPU do *host*) ou, a nível de instrução, em conjuto com o MPSIM que emula instruções do microcontrolador TI MSP430. Porém, o Cooja também permite simular programas escritos para outros sistemas operacionais (como

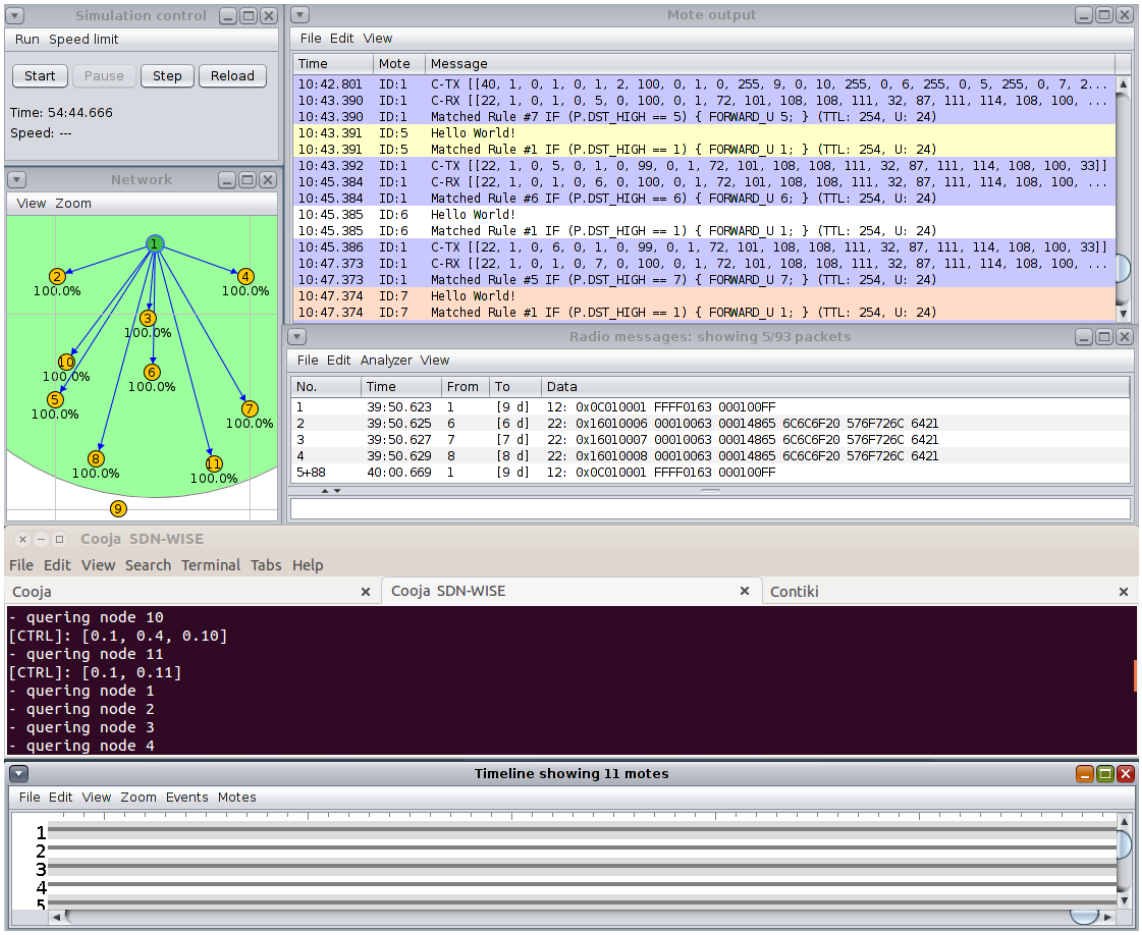


Figura 21 – Visão geral do simulador de redes Cooja.

o Tiny OS), além de possibilitar que módulos sejam criados puramente em Java, em nível de aplicação. Esta última abordagem tem como principal vantagem a agilidade de desenvolvimento sendo uso aplicado ao processo de criação e teste de algoritmos (OSTERLIND et al., 2006).

O Cooja disponibiliza várias ferramentas que auxiliam na depuração do código e na criação de cenários com módulos sensores simulados. A Figura 21, mostra algumas das principais ferramentas. A *Network* permite a visualização da troca de pacotes e o posicionamento dos módulos de forma gráfica. Já as ferramentas *Mote Output* e *Radio messages* registram, respectivamente, os *logs* publicados pelos módulos e as mensagens transmitidas através dos seus rádios. Na *Timeline* é possível observar as transmissões realizadas. Por fim, o Cooja oferece uma interface *Transmission Control Protocol* (TCP) que pode ser acessada através de um terminal externo, o qual neste trabalho permite a integração do controlador com os módulos simulados.

5.1.2 Nodos SDN-WISE

Os códigos dos módulos sensores sorvedouro e nodo, escritos em Java e disponibilizados pelos criadores do SDN-WISE para testes e desenvolvimento da comunidade acadêmica foram utilizados como base no desenvolvimento e simulações apresentadas neste trabalho. O funcionamento destes módulos segue o padrão estabelecido pelo SDN-WISE e serão descritas a seguir.

Assim sendo, a rede é inicializada pelo sorvedouro, que primeiramente verifica se há um controlador disponível para conexão. Caso exista, o sorvedouro se conecta e passa a transmitir requisições e dados da rede para o controlador e no caminho inverso, receber regras de roteamento além de pacotes de configuração e dados. Em um segundo momento, o sorvedouro utiliza o mecanismo de descoberta da topologia (descrito na Subseção 2.3.3) para que os módulos da rede encontrem o vizinho que represente o melhor salto em direção ao sorvedouro, pois será através desse vizinho que os pacotes de *Report* serão enviados para o controlador. Portanto, como pode ser visto na Figura 22 um pacote do tipo *RegProxy*, é enviado do sorvedouro para o controlador, através de uma conexão TCP. Este pacote notifica o controlador da existência de um sorvedouro na rede e possui os dez bytes do cabeçalho, como descrito na Subseção 2.3.2, e no payload os seguintes campos:

- *Datapath Identifier* (DPID) - *Bytes* 10-17: DPID do sorvedouro.
- MAC - *Bytes* 18-23: MAC do sorvedouro.
- *Port* - *Bytes* 24-31: Porta física do sorvedouro.
- *Internet Protocol address* (IP) - *Bytes* 32-35: Endereço IP do sorvedouro.
- TCP - *Bytes* 36-37: Porta TCP do sorvedouro.

Time	Mote	Message
00:00.000	ID:1	C-TX [[38, 1, 0, 1, 0, 1, 7, 100, 0, 1, 109, 97, 104, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 127, 0, 0, 1, 0, 0]]

Figura 22 – Pacote *Regproxy* enviado do sorvedouro para o controlador. Em base decimal.

Após verificar a disponibilidade e se conectar ao controlador, o sorvedouro inicia o processo de descoberta da topologia da rede. Para isso, ele envia através de *broadcast*, um *Beacon* que possui apenas dois *bytes* de *payload* anexados aos dez *bytes* do cabeçalho padrão do SDN-WISE. Como pode ser visto da Figura 23, o primeiro *byte* de *payload* representa a distância, em números de saltos, até o sorvedouro e o segundo byte representa a bateria do remetente. Ao receber esse *beacon*, o módulo verifica se o valor da

distância representado no primeiro *byte* é menor que o mínimo histórico de valores de distância recebidos. Caso seja, ele atualiza o próximo salto em direção ao sorvedouro com o endereço do remetente do pacote, caso contrário ele ignora a informação. Por fim, se um novo endereço para o próximo salto foi encontrado, o módulo incrementa o valor do primeiro byte de um, insere seu nível de bateria no segundo byte e reenvia o pacote por *broadcast*. Esse processo de envio de *Beacons* é repetido periodicamente a uma taxa configurável pelo controlador.

No.	Time	From	To	Data
1	00:10.053	1	[3 d]	12: 0x0C010001 FFFF0163 000100FF
2	00:19.088	10	[6 d]	12: 0x0C01000A FFFF0163 000101FE
3	00:19.093	2	[4 d]	12: 0x0C010002 FFFF0163 000101FE
4	00:20.095	6	[3 d]	12: 0x0C010006 FFFF0163 000101FE
5	00:20.105	1	[3 d]	12: 0x0C010001 FFFF0163 000100FF
6	00:28.131	7	[4 d]	12: 0x0C010007 FFFF0163 000102FE
7	00:28.132	3	[3 d]	12: 0x0C010003 FFFF0163 000102FE
8	00:29.119	14	2,3	12: 0x0C01000E FFFF0163 000102FE
9	00:29.124	11	[6 d]	12: 0x0C01000B FFFF0163 000102FE

Figura 23 – Pacotes de *Beacons* utilizados para descoberta do melhor caminho até o sorvedouro. Em base hexadecimal.

Após encontrar o próximo salto para chegar até o sorvedouro mais próximo os nodos iniciam o envio dos pacotes de *Report* para o controlador. O tamanho deste pacote é variável, quanto mais vizinhos um nodo possui, maior será o *payload*. Dessa forma, além dos dez *bytes* fixos para o cabeçalho, o *Report* apresenta a seguinte estrutura:

- *Distance* - *Byte* 10: Distância, em saltos, que o módulo está do sorvedouro;
- *Battery* - *Byte* 11: Informação do nível da bateria do módulo. Sendo que, *0xFF* é o nível mais alto e *0x00* indica que a bateria está esgotada;
- *NeigborsSize* - *Byte* 12: Indica a quantidade de módulos vizinhos sendo reportados;
- *NeighborAddress 1* - *Bytes* 13-14: Endereço do primeiro vizinho da lista;
- *LinkQuality 1* - *Byte* 15: indicador RSSI de qualidade do enlace entre o módulo e o seu primeiro vizinho da lista;
- *NeighborAddress n* - *Bytes* ...-... : Endereço do n-ésimo vizinho da lista;
- *LinkQuality n* - *Byte* ... : indicador RSSI de qualidade do enlace entre o módulo e o seu n-ésimo vizinho da lista;

Time	Mote	Message
08:21.243	ID:1	C-TX [[19, 1, 0, 14, 0, 1, 2, 97, 0, 1, 2, 230, 2, 0, 2, 255, 0, 3, 255]]

Figura 24 – Pacote de *Report* utilizado para informar a topologia da rede. Em base decimal.

Na Figura 24, é possível ver um exemplo de *Report* do módulo que possui endereço 14 da rede simulada e mostrada na Figura 25. O primeiro *byte* do payload mostra que bateria dele está com um nível 230 (25 abaixo do valor máximo de 255). Ele possui dois vizinhos em seu raio de cobertura (circulo verde), que possuem endereço 2 e 3. A RSSI de todos os módulos dentro do raio de cobertura é considerado máximo (255) na simulação.

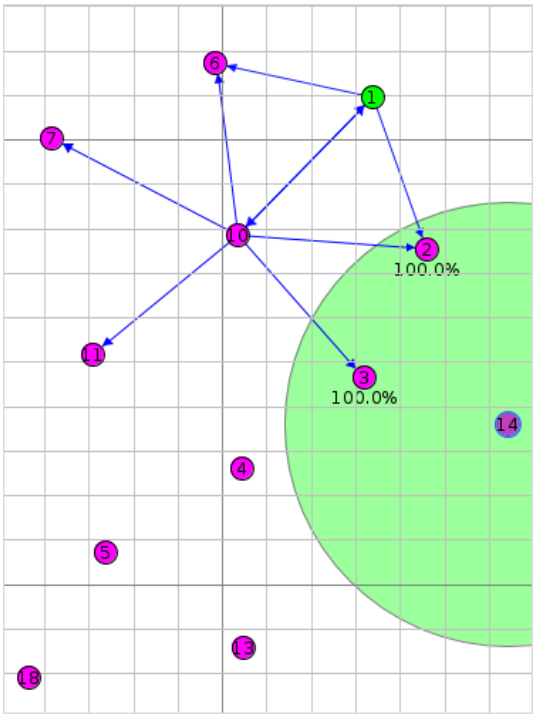


Figura 25 – Rede simulada no Cooja.

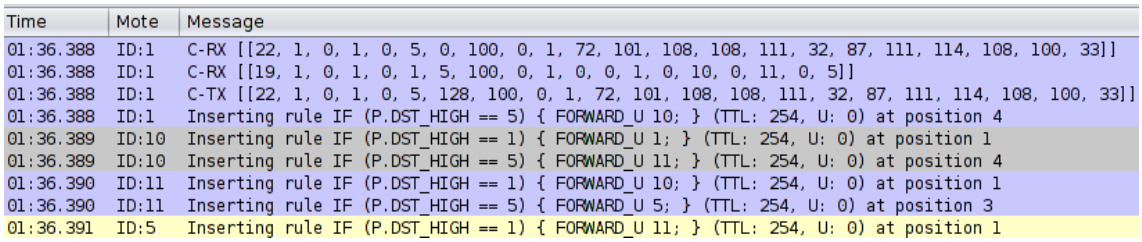
5.1.3 Controlador SDN-WISE

A linguagem de desenvolvimento do controlador SDN fica a critério do desenvolvedor, uma vez que basta implementar a API do SDN-WISE para que a comunicação com a rede seja realizada. Os criadores do SDN-WISE também disponibilizam uma implementação básica de um controlador que utiliza o algoritmo de Djikstra para encontrar o caminho mais curto (menor número de saltos) entre um módulo fonte e um sorvedouro.

O controlador é executado como um servidor portanto na inicialização do seu processo uma conexão é aberta para que um sorvedouro se conecte. Para isso, um endereço e porta IP são definidos em um arquivo *config.ini*, que também contem outras informações como tipo de algoritmo e tamanho máximo do *payload* a ser transmitido. Ao receber uma conexão o controlador passa a enviar, periodicamente, pacotes de dados (Figura 26) para a rede. Ao receber um destes pacotes, o sorvedouro deve verificar se há alguma regra de correspondência em sua *WISE Flow Table* que permita o processamento do dado. Caso exista, a ação correspondente é utilizada, caso contrário o sorvedouro requisita uma nova regra. Para isso, o SDN-WISE define que uma *Request*, contendo os seguintes campos em seu *payload*, deve ser enviada:

- *id* - *Byte* 10: Identificador da requisição;
- *Part* - *Byte* 11: Devido ao encapsulamento do pacote recebido, a *Request* pode ser divida em partes, quando necessário. Portanto, este campo identifica qual parte do pacote está sendo transmitido;
- *Total* - *Byte* 12: Indica o número total de parte do pacote;
- *Unmatched Packet* - *Bytes 13-...*: Pacote, sem regra de correspondência, recém-recebido.

A implementação Java do sorvedouro, disponibilizada pelos criadores do SDN-WISE envia um pacote fora do padrão para realizar a requisição. O pacote possui o campo *Type* do cabeçalho definido como 128 e envia apenas os *bytes* referentes ao campo *Unmatched Packet* em seu *payload*, como pode ser



Time	Mote	Message
01:36.388	ID:1	C-RX [[22, 1, 0, 1, 0, 5, 0, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
01:36.388	ID:1	C-RX [[19, 1, 0, 1, 0, 1, 5, 100, 0, 1, 0, 0, 1, 0, 10, 0, 11, 0, 5]]
01:36.388	ID:1	C-TX [[22, 1, 0, 1, 0, 5, 128, 100, 0, 1, 72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]]
01:36.388	ID:1	Inserting rule IF (P.DST_HIGH == 5) { FORWARD_U 10; } (TTL: 254, U: 0) at position 4
01:36.389	ID:10	Inserting rule IF (P.DST_HIGH == 1) { FORWARD_U 1; } (TTL: 254, U: 0) at position 1
01:36.389	ID:10	Inserting rule IF (P.DST_HIGH == 5) { FORWARD_U 11; } (TTL: 254, U: 0) at position 4
01:36.390	ID:11	Inserting rule IF (P.DST_HIGH == 1) { FORWARD_U 10; } (TTL: 254, U: 0) at position 1
01:36.390	ID:11	Inserting rule IF (P.DST_HIGH == 5) { FORWARD_U 5; } (TTL: 254, U: 0) at position 3
01:36.391	ID:5	Inserting rule IF (P.DST_HIGH == 1) { FORWARD_U 11; } (TTL: 254, U: 0) at position 1

Figura 26 – Troca de mensagens entre o controlador e o sorvedouro para estabelecer uma rota até o nodo 5.

visualizado na Figura 26. O controlador ao receber o pacote e reconhecê-lo como uma requisição, mesmo que fora de padrão, inicia o algoritmo de Djikstra e imediatamente envia para o sorvedouro um pacote do tipo *OpenPath*, que consiste dos seguintes campos:

- *Header* - *Bytes* 0-9: Cabeçalho do *Wise Packet*;
- *WindowSize* - *Byte* 10: As regras de correspondência podem ser criadas através de *Windows*, que são compostas por dois valores e um operador relacional. O *Byte* WindowSize indica quantas *Windows* serão criadas;
- *Window 1* - *Bytes* 11-15: A primeira *Window* adicionada às regras;
- *Window n* - *Bytes* ... - ... : A n-ésima *Window* adicionada às regras;
- *Address 1* - *Bytes* ... - ...: O primeiro módulo no caminho;
- *Address k* - *Bytes* ... - ...: O k-ésimo módulo no caminho.

O controlador anexa o endereço de cada módulo no caminho encontrado pelo algoritmo nos últimos *bytes* do pacote *OpenPath*. A vantagem de utilizar esta abordagem é reduzir o número de pacotes enviados pelo controlador para criar um caminho de roteamento na rede, já que não será necessário configurar a *WISE flow table* de cada módulo individualmente. Assim sendo, ao receber o pacote, cada módulo cria uma regra de correspondência para alcançar o primeiro e o ultimo módulo do caminho, através dos módulos imediatamente anterior e posterior a sua posição no caminho. Na Figura 26 é possível observar que ao requisitar um caminho do sorvedouro 1 até o nodo 5 (ver Figura 25), o controlador responde com um caminho através dos módulos 10 e 11. Este processo é repetido até que o sorvedouro possua um caminho de roteamento para todos os nodos da rede.

5.2 Pacotes

Dois novos pacotes foram criados para auxiliar na configuração e eficiência energética da rede. O primeiro pacote, batizado de *SOURCE_SETUP*, estende a funcionalidade do *Config* e tem como objetivo configurar os nodos fontes da rede. Dessa forma, ao receber o *SOURCE_SETUP*, o módulo entende que ele deve gerar dados e enviá-los para o controlador, através de um sorvedouro. No *payload* são anexados dois *bytes* que indicam o tamanho dos pacotes a serem gerados e mais dois *bytes* para informar o perfil de geração dos dados. Assim sendo, o pacote de configuração possui os seguintes campos:

- *Header* - *Bytes* 0-9: Cabeçalho do *Wise Packet*;
- *ConfigId* - *Byte* 10: Indica se a configuração será escrita ou lida do nodo. Este campo é composto por um *bit* que indica se o controlador deseja gravar ou receber o estado da configuração do módulo e

um ID que indica qual configuração o controlador se refere. Como o protocolo SDN-WISE já define 17 tipos de configuração o ID criado para o novo pacote é 18;

- *Params* - *Bytes* 11- ... : Opcional nos pacotes de configuração do SDN-WISE, esse campo é utilizado para anexar dois *bytes* referentes ao perfil de geração além de dois *bytes* que indicam o tamanho do dado a ser gerado.

O segundo pacote, que possui ID 13, foi batizado de *ClearPath* e tem como objetivo desativar um caminho com apenas uma mensagem enviada do controlador para a rede. O SDN-WISE prevê a exclusão de regras de correspondências das *WISE flow table* dos módulos, porém isto é feito individualmente o que diminui a eficiência energética da rede. Por isso, utilizando-se do princípio empregado no *OpenPath* neste pacote são anexados os IDs de todos os módulos do caminho ativo:

- *Header* - *Bytes* 0-9: Cabeçalho do *Wise Packet*.
- *Address 1* - *Bytes* ... - ...: O primeiro módulo no caminho.
- *Address k* - *Bytes* ... - ...: O k-ésimo módulo no caminho.

A razão para a criação deste pacote é aumentar o grau de liberdade do controlador ao redefinir caminhos em tempo real. Por exemplo, se existe um caminho ativo porém o controlador percebe que esse caminho já não é o mais adequado segundo seu algoritmo, ele deve retirar as regras de correspondências dos módulos que participam do caminho atual e reconfigurar a rede com uma nova rota.

Visando disponibilizar os pacotes descritos nesta Seção para trabalhos futuros foram criados métodos no código da classe abstrata do controlador. Assim sendo, na linha 10 do Algoritmo 5.1 foi definida a função *sendSourceConfig* para enviar o *SOURCE_SETUP* e na linha 31 foi definida a função sendClearFlowtable que cria um pacote do tipo *OpenPath*, porém modifica o tipo do pacote para 13 e não adiciona nenhum *Window*.

Código 5.1 – Implementação em Java dos pacote de configuração *SOURCE_SETUP* e do *ClearPath*.

```
1
2   /*   sendSourceConfig - Utilizado para enviar SOURCE_SETUP para o modulo source.
3    *   @param netId Identificador da rede.
4    *   @param destination endereco do destinatario. Modulo a ser configurado como source.
5    *   @param distribution Perfil de geracao.
6    *   @param dataSize Tamanho do pacote a ser gerado pelo source.
7    *   @return void
8    */
9
10   public final void sendSourceConfig(int netId, NodeAddress destination, String distribution, int
      dataSize) {
11       byte d = 0;
12       ConfigPacket cp = new ConfigPacket(netId, sinkAddress, destination);
13
14       if(distribution.equals("CBR"))
15           d = 1;
16
17       byte pl[] = {(byte)(18 | (1<<7)), 0, d, (byte) (dataSize >> 8), (byte) dataSize};
18       cp.setPayload(pl)
19           .setNxhop(sinkAddress);
20       sendNetworkPacket(cp);
21   }
22
```

```
23
24    /*   sendClearFlowtable - Utilizado para enviar ClearPath para os pertencentes ao caminho
       especificado no @param path.
25     *
26     *   @param netId Identificador da rede.
27     *   @param destination endereco do destinatario. Sink da rede que integra o @param path.
28     *   @param path Lista com todos os modulos que integram o caminho a ser desfeito.
29     *   @return void
30     */
31    public final void sendClearFlowtable(byte netId, NodeAddress destination, List<NodeAddress> path)
       {
32        OpenPathPacket op = new OpenPathPacket(netId, sinkAddress, destination);
33        //Como o pacote segue o mesmo padrao do OpenPath, o codigo foi reutilizado. Porem o tipo do
       pacote foi modificado para 13.
34        op.setPath(path)
35          .setNxhop(sinkAddress)
36          .setType((byte)13);
37
38        sendNetworkPacket(op);
39    }
```

5.3 Nodos Simulados

Os módulos sensores sorvedouro e nodo da camada de dados utilizam o código disponibilizado pelos criadores do SDN-WISE com algumas adaptações realizadas para este trabalho. A primeira modificação notável é a adoção de dois estados de funcionamento, um que indica modo *Normal* e um segundo modo para *Economia de Energia*. Ao primeiro modo, atribuiu-se o comportamento padrão de envio de pacotes para o controlador, em que os módulos enviam um pacote de *Beacon* a cada dez segundos e um de *Report* a cada vinte segundos. O modo *Economia de Energia* diminui pela metade a frequência de envios destes pacotes de controle, sendo que a mudança de um estado para outro ocorre sem interferência do controlador.

Na inicialização do seu processo, cada módulo aloca um contador para monitorar quantos pacotes de *Report* foram enviados ao controlador, como pode ser visto na linha 63 do código 5.2 se o módulo verificar que foram enviados quatro pacotes de *Report* e nenhum anúncio de caminho foi recebido, então ele muda seu estado para *Economia de Energia*. Isto não só contribui para a economia de energia do módulo que entra neste modo, mas também diminui a quantidade de pacotes que seus vizinhos recebem, e portanto reduz o consumo energético da vizinhança.

Código 5.2 – Transição entre estado normal e de economia de energia.

```
41    //Indicam o valor padrao do periodo de envio dos Beacons e Reports, respectivamente.
42    public final static byte SDN_WISE_DFLT_CNT_BEACON_MAX = 10;
43    public final static byte SDN_WISE_DFLT_CNT_REPORT_MAX = 2 * SDN_WISE_DFLT_CNT_BEACON_MAX;
44
45    //Indicam novos periodos de envio dos Beacons e Reports, respectivamente.
46    //Valores configuraveis pelo controlador SDN.
47    private int SDN_WISE_NEW_CNT_BEACON_MAX = 0;
48    private int SDN_WISE_NEW_CNT_REPORT_MAX = 0;
49
50    //Armazenam o valor atual dos periodos de envio dos Beacons e Reports, respectivamente.
51    int cnt_beacon_max = SDN_WISE_DFLT_CNT_BEACON_MAX;
52    int cnt_report_max = SDN_WISE_DFLT_CNT_REPORT_MAX;
53
54    //Indica se o modulo integra um caminho (true) ou nao (false).
55    private boolean pathActive = false;
```

```
56
57      //Contador de quantos Report foram enviados.
58      private int cntReportSent = 0;
59
60      //Indica se o modulo esta no modo normal (true) ou economia de energia(false).
61      private boolean NORMAL_STATE = true;
62
63      if( ++cntReportSent == 4 && !pathActive ) {
64          NORMAL_STATE = false;
65
66          if(SDN_WISE_NEW_CNT_REPORT_MAX != 0)
67              cnt_report_max = SDN_WISE_NEW_CNT_REPORT_MAX*2;
68          else
69              cnt_report_max = SDN_WISE_DFLT_CNT_REPORT_MAX*2;
70
71          if(SDN_WISE_NEW_CNT_BEACON_MAX != 0)
72              cnt_beacon_max = SDN_WISE_NEW_CNT_BEACON_MAX*2;
73          else
74              cnt_beacon_max = SDN_WISE_DFLT_CNT_BEACON_MAX*2;
75      }
```

Caso uma mensagem de *OpenPath* for recebida indicando que o nodo foi adicionado à um caminho o estado é alterado para *Normal*, ou seja, os períodos de envio dos *Beacons* e *Reports* voltam ao padrão. Caso um pacote *ClearPath* (detalhado na Subseção 5.2) que indica a retirada do módulo de um caminho for recebida o estado é alterado para *Economia de Energia*.

As outras alterações realizadas no código dos módulos tiveram como propósito a adequação da camada de rede aos novos pacotes definidos na Subseção 5.2. Ao receber o pacote de configuração do tipo *SOURCE_SETUP*, o módulo salva as informações referentes ao tamanho dos dados à serem enviados, o perfil de geração e passa a enviar pacotes de dados para o controlador, através do sorvedouro da rede. Quando um pacote do tipo *CLEAR_PATH* é recebido, os módulos devem excluir até duas regras de correspondência de suas respectivas *WISE flow table*. Para isso, os módulos verificam a sua posição no caminho recebido no pacote. Se eles estão entre dois módulos, então duas regras devem ser excluídas, caso contrário apenas uma regra é excluída. Em seguida, o módulo utiliza repassa o *CLEAR_PATH* pra o próximo módulo do caminho.

5.4 Controlador

O projeto do controlador disponibilizado pelos criadores do SDN-WISE possui classes abstratas que implementam funções básicas como o envio de mensagens para a rede, identificação do tipo de pacote recebido e criação de um grafo da rede. Com isso, neste trabalho criou-se um novo controlador intitulado ControllerTCC que pode executar até três algoritmos de roteamento de maneira não simultânea, sendo eles o algoritmo de Dijkstra herdado do controlador SDN-WISE e dois outros algoritmos que visam estender o custo de vida da rede: caminhos disjuntos e de recompensas negativas. Além do mais o ControllerTCC monitora os caminhos da rede a cada sessenta segundos para verificar se o caminho atual é o melhor, segundo o algoritmo em execução.

Na inicialização do controlador um arquivo de configuração *config.ini* como o mostrado no Código 5.3 deve ser lido. Na seção *lower* do arquivo é possível obter a configuração de endereço e porta IP além de outros parâmetros utilizados na conexão entre o controlador e a camada de dados. Na seção *map*, é possível definir o tempo máximo que um módulo pode ficar sem enviar *Report* para o controlador antes de ser excluído do grafo, um valor para a máxima variação de RSSI admitida e se o grafo será plotado

em uma interface gráfica ou apenas armazenado para ser utilizado pelo algoritmo de roteamento. Por fim, a seção *algorithm* contém os parâmetros utilizados para configurar de forma estática o algoritmo e os módulos da rede. Por exemplo, no Código 5.3, o campo *TYPE* define que o algoritmo a ser utilizado é o de recompensas negativas e o campo *CHECK_INTERVAL* define o intervalo (em milissegundos) entre execuções de um mecanismo do controlador que verifica se os caminhos ativos na rede ainda são considerados os melhores, de acordo com o algoritmo em uso. Os demais campos são utilizado para configurar os nodos fonte da rede, neste caso os IDs dos nodos fonte são 0.17 e 0.18, o perfil de geração dos dados é o Constant bit rate (CBR) e o tamanho dos pacotes gerados pelo nodo fonte deve ser de 50 *bytes*.

Código 5.3 – Arquivo config.ini de configuração do controlador

```
{
  "Controller": {
    "lower": {
      "PORT": "9991",
      "IP": "localhost",
      "MAX_PAYLOAD": "4096",
      "TYPE": "TCP",
      "IS_SERVER":"TRUE"
    },
    "algorithm": {
      "TYPE": "TCC_Negative_Reward",
      "CHECK_INTERVAL":"60000",
      "SOURCE_NODE_ID" : "0.17 0.18",
      "DISTRIBUTION" : "CBR",
      "DATA_SIZE_BYTES" : "50"
    },
    "map": {
      "TIMEOUT": "90",
      "RSSI_RESOLUTION": "20",
      "GRAPH":"GUI"
    }
  }
}
```

A cada recepção de um *Report* de um dos módulos designados como fonte um pacote de *SOURCE_SETUP* é enviado como resposta até que um pacote de dados do nodo fonte seja recebido, confirmando que o nodo em questão está configurado corretamente.

O GraphStream[1] é uma biblioteca publica para projetos em Java que permite modelar e analisar grafos dinamicamente. Ela é utilizada pelo SDN-WISE para plotar a topologia da rede sendo simulada (Figura 27). Dessa forma, quando mensagens de *Report* são recebidas pelo controlador um grafo onde cada vértice representa um módulo e as arestas unidirecionais representam a comunicação entre os módulos vizinhos é criado. Na implementação do controlador Djiskstra original do SDN-WISE as arestas possuem peso 1 representando um salto independentemente da distância, RSSI do enlace ou nível energético dos módulos. No ControllerTCC este comportamento é mantido para o djikstra e estendido para o algoritmo de caminhos disjuntos, porém no de recompensas negativas o peso utilizado é o somatório das energias residuais dos dois vértices conectados por uma aresta multiplicado por dois a cada uso do enlace. Para isso, o nível energético mais recente de cada módulo passou a ser armazenado em uma tabela *hash* que é então repassada para o algoritmo de recompensas negativas e também para uma nova função que atualiza, em tempo real, os pesos de cada aresta na grafo mostrado na interface gráfica.

[1] http://graphstream-project.org/

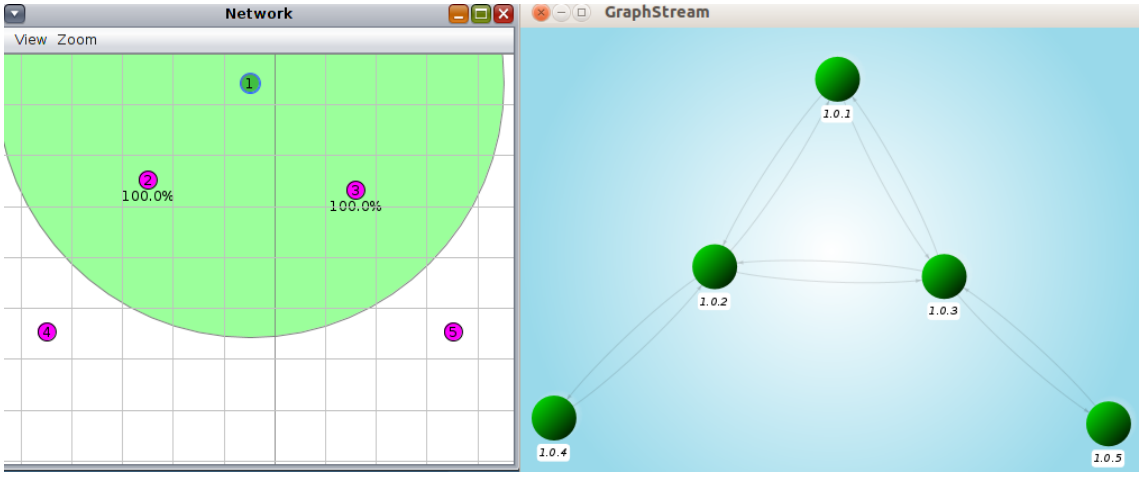


Figura 27 – Ambiente de simulação no Cooja à esquerda e grafo plotado pelo controlador à direita.

5.5 Algoritmos

Os algoritmos de roteamento propostos neste trabalho têm como objetivo estender o tempo de vida da RSSF. Para isso o primeiro algoritmo utiliza a abordagem de caminhos disjuntos com o intuito de limitar o consumo energético dos nodos em um caminho e o segundo utiliza recompensas negativas para penalizar os enlaces que possuem pior nível energético residual ou que estejam em uso. Para efeito de comparação, o algoritmo de Djikstra original do SDN-WISE foi mantido na implementação do controllerTCC.

5.5.1 Djikstra

O algoritmo de Djikstra é executado para encontrar o menor caminho considerando o número de saltos entre o nodo fonte e o sorvedouro. Como ocorre no controlador SDN-WISE, o caminho encontrado será mantido fixo ao longo da simulação uma vez que neste algoritmo não existe nenhum mecanismo de atualização da rede.

Algoritmo 1: DJIKSTRA

Entrada: $NetGraph, PathRequestPacket$
Saída: Caminho de roteamento

```
1 início
2 |  Djikstra.init(NetGraph)
3 |  Path ← Djikstra.getPath(PathRequestPacket.src, PathRequestPacket.dst)
4 |  retorna Path
5 fim
```

Na linha 2 do Algoritmo 1 um objeto Djikstra é instanciado com uma referência para o grafo gerado a partir dos *Reports* recebidos, em seguida um único caminho de roteamento entre os módulos sorvedouro *(PathRequestPacket.src)* e fonte *(PathRequestPacket.dst)* é encontrado.

5.5.2 Caminhos Disjuntos

Este algoritmo utiliza caminhos disjuntos para encontrar rotas entre os nodos fontes e o sorvedouro de uma RSSF. Assim sendo, neste algoritmo todos os caminhos possíveis e disjuntos entre os dois módulos de interesse são encontrados utilizando como métrica o menor número de salto, em seguida verifica-se que todos os nodos possuem nível energético acima de um limiar mínimo exigido e finalmente escolhe-se

Módulo	Nível energético
1	255
2	250
3	3
4	100
5	100
6	255

Tabela 2 – Exemplo: Tabela de níveis energéticos.

o caminho que possua o maior somatório das energias residuais de seus módulos. Por exemplo, um controlador que foi configurado com um limiar mínimo igual à 5 encontra dois caminhos A e B, compostos pelos módulos [1, 2, 3, 6] e [1, 4, 5, 6], respectivamente. Todos os módulos possuem seus níveis energéticos mostrados na tabela 2 sendo que o módulo 1 é o sorvedouro e 6 o fonte. Assim sendo, o caminho escolhido como melhor será o B, pois apesar de A possuir o maior somatório dos níveis energéticos de seus módulos (763) o módulo 3 possui nível energético abaixo do limiar mínimo.

O pseudo código do algoritmo de caminhos disjuntos pode ser separado em duas etapas lógicas. Sendo assim o Algoritmo 2 mostra a primeira etapa do processo que encontra todos os possíveis caminhos disjuntos, enquanto o Algoritmo 3 mostra a segunda etapa que envolve o processo de escolha do melhor caminho considerando o nível energético dos módulos. Esta ultima etapa do algoritmo é executada pelo controllerTCC a cada intervalo de tempo definido em *CHECK_INTERVAL* para verificar se o caminho ativo ainda é considerado o melhor.

Algoritmo 2: *Disjoint Path - Etapa 1*

```
   Entrada: NetGraph, PathRequestPacket
   Saída: Caminho de roteamento
 1 início
 2 |  Djikstra.init(NetGraph)
 3 |  Path ← Djikstra.getPath(PathRequestPacket.src, PathRequestPacket.dst)
 4 |  se Path encontrado então
 5 |  |  PathsFound.push(Path);
 6 |  |  para cada node ∈ Path faça
 7 |  |  |  se node.address != sink e node.address != source então
 8 |  |  |  |  Remove módulo do grafo NetGraph.remove(node);
 9 |  |  |  senão
10 |  |  |  |  Mantém módulo no Grafo
11 |  |  |  fim
12 |  |  fim
13 |  |  goto início
14 |  senão
15 |  |  Executa etapa 2
16 |  |  retorna resultado da etapa 2
17 |  fim
18 fim
```

Na linha 3 do Algoritmo 2 um objeto do tipo Djikstra é utilizado para encontrar o melhor caminho (com menor número de saltos). Na linha 8 os módulos do caminho encontrado são removidos do grafo da rede pois, com exceção dos módulos fonte e sorvedouro, estes não poderão mais ser utilizado em outro caminho. Por fim na linha 15 a segunda etapa do algoritmo de caminhos disjuntos é executada.

A segunda etapa pode ser executada uma vez para encontrar um novo caminho e periodicamente para verificar se todos os módulos do caminho ativo ainda possuem seu nível energético acima do limiar mínimo, definido no controllerTCC. Portanto, na linha 4 do Algoritmo 3 o nível energético da bateria dos

módulos de cada caminho encontrado é verificado, se um dos nodos de um caminho está abaixo do limiar mínimo exigido então o caminho é retirado da lista de possíveis rotas entre os nodos fontes e sorvedouro (linha 8). Em seguida, com todos os caminhos verificados somente aquele com menor número de saltos será retornado. Por fim, é válido notar que esta abordagem mantém a escolha de um caminho até que um dos módulos ativos tenha seu nível de energia reduzido abaixo do limiar mínimo.

Algoritmo 3: *Disjoint Path - Etapa 2*

```
   Entrada: PathsFound, PathRequestPacket
   Saída: Caminho de roteamento
 1 início
 2   para cada Path ∈ PathsFound faça
 3     para cada node ∈ Path faça
 4       se node.bateria > limiar então
 5         Path é utilizável
 6       senão
 7         Path não é utilizavel
 8         PathsFound.remove(Path);
 9       fim
10     fim
11   fim
12   bestPath ← PathsFound.shortestPath()
13   se bestPath encontrado então
14     retorna bestPath
15   senão
16     retorna null
17   fim
18 fim
```

5.5.3 Recompensas Negativas

No algoritmo de recompensas negativas os caminhos são atualizados sempre que necessário de modo que a rota em uso sempre é aquela que possui melhor nível energético residual em seus módulos. A vantagem deste algoritmo em relação aos anteriores é portanto o consumo distribuído da energia na rede em contraste com o uso constante de um único caminho até que o nível energético de um dos módulos dele seja igual ou próximo à zero.

Nas linhas 3 e 4 do Algoritmo 4 é calulado o peso de cada aresta do grafo como o máximo valor possível de energia residual (Max_{Enr}) de cada módulo multiplicado por dois menos o somatório da energia residual (Enr) atual dos nodos em cada ponta de uma aresta. Sendo que a subtração é feita para que os enlaces com maior nível energético residual possuam o menor peso. Por fim, multiplica-se o peso por 2^N, onde N é a quantidade de caminhos que utilizam o enlace. Por exemplo, uma rede composta pelos módulos descritos na Tabela 2 tem seus enlaces representados como E[M1, M2], onde E é o enlace, M1 e M2 representam os módulos e o Max_{Enr} de um módulo é 255. Sendo assim, se a rede possui possui um enlace A[1, 2] não utilizado, um enlace B[2, 4] utilizado no caminho W e um terceiro enlace C[5, 6] utilizado nos caminhos X, Y e Z então os pesos calculados de acordo com a equação 5.1 serão 5, 320 e 1240 para os enlaces A, B e C, respectivamente.

$$weight = (2 * Max_{Enr} - Enr0 + Enr1) * (2^n) \tag{5.1}$$

Ao encontrar um caminho, os pesos atribuídos aos enlaces que o compõem são dobrados, fazendo com que os caminhos encontrados nas demais iterações do algoritmo distribuam o consumo energético na

rede entre os diversos nodos que a compõem.

Algoritmo 4: Recompensas Negativas

Entrada: *input* : $PathRequestPacket, NetGraph, EnergyMap, ActivePaths$
Saída: Caminhos de roteamento

```
1  início
2      para cada arc ∈ NetGraph faça
3          weight = Max_Enr*2 - (EnergyMap(arc.node0) + EnergyMap(arc.node1));
4          weight *= 2^(arc.QtyOfPathsArcIsOn)
5          NetGraph.SetCost(arc, weight*NetGraph.getCost(arc))
6      fim
7      Djikstra.init(NetGraph)
8      enquanto
        Path ← Djikstra.getPath(PathRequestPacket.src, PathRequestPacket.dst) != null
       faça
9          PathsFound.push(Path);
10         para cada node ∈ Path e node! = Sink faça
11             NetGraph.SetCost(node,Path.next(node),2*Djikstra.getCost(node,Path.next(node))
12         fim
13         Djikstra.update(NetGraph)
14     fim
15     retorna PathsFound;
16 fim
```

Assim como acontece no *caminhos disjuntos*, o controllerTCC verifica se os caminhos escolhidos possuem todos os seus módulos com nível energético acima de um limiar mínimo definido para a rede e, periodicamente executa o algoritmo 4 para assegurar que os caminhos escolhidos permanecem como a melhor escolha, de acordo com o recompensas negativas.

5.6 Avaliação dos Algoritmos

Para verificar a efetividade dos algoritmos de roteamento descritos na Seção 5.5 em estender o tempo de vida de uma RSSF uma rede com 18 nodos foi criada e sobre ela dois cenário de testes foram pensados. No primeiro cenário há um nodo que atua como fonte (18) e um outro como sorvedouro (1) e no segundo cenário existem dois nodos fontes (17 e 18) e um sorvedouro (1). Para a simulação, assume-se que os nodos escolhidos como fonte e sorvedouro possuem baterias com capacidade muito acima do restante da rede e por isso elas serão desconsideradas nos resultados. Assim sendo, o percentual de energia drenada das baterias do restante dos módulos que apenas roteiam dados é de $0,002\%$ a cada segundo representando o consumo de processamento do nodo, 0.006% à cada transmissão e 0.01% na recepção de pacotes.

A Figura 28 mostra como os módulos foram distribuídos manualmente no Cooja de tal forma que todos eles possuam ao menos um vizinho em seu raio de cobertura. Além do mais, é possível observar que a rede foi espelhada a partir de um eixo central formado pelos nodos 1, 10, 11 e 12 dividindo-a em dois lados que serão disponibilizados igualmente entre os nodos 17 e 18 no segundo cenário. O meio de transmissão escolhido no Cooja foi o *Unit Disk Graph Model* (UDGM) que permite escolher um raio de alcance que delimita a cobertura para transmissão e recepção dos nodos, um raio de interferência em que um nodo detectará uma transmissão mas não a receberá corretamente e além de definir as probabilidades de recepção e transmissão realizadas com sucesso na rede. Com isso, definiu-se que na rede simulada não há perdas por interferência, todos os pacotes têm probabilidades máximas de serem entregues e todos os vizinhos ao alcance de um módulo possuem o mesmo RSSI (255), ou seja, a distância física entre dois nodos não é considerada nas simulações.

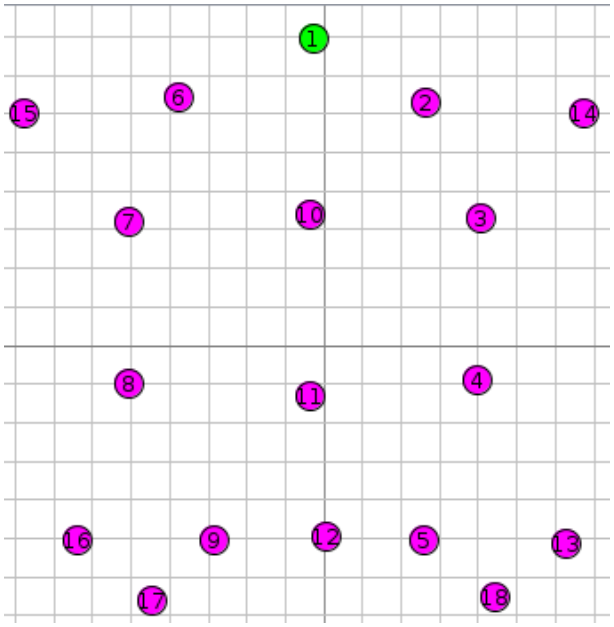


Figura 28 – Ambiente de simulação

O controlador foi executado em um computador equipado com processador Intel Core I7 e 6 *Gigabytes* de memória RAM. A configuração foi realizada de acordo com arquivo config.ini mostrado no Código 5.4, onde a variável *SOURCE_NODE_ID* é alterada para ”0.18” no primeiro cenário e o *TYPE* é alternado entre DIJKSTRA, TCC_Disjoint_Path e TCC_Negative_Reward para executar os algoritmos de Djikstra, caminhos disjuntos e recompensas negativas, respectivamente.

Código 5.4 – Arquivo config.ini de configuração do controlador utilizado na simulação

```
{
  "Controller": {
    "lower": {
      "PORT": "9991",
      "IP": "localhost",
      "MAX_PAYLOAD": "4096",
      "TYPE": "TCP",
      "IS_SERVER":"TRUE"
    },
    "algorithm": {
      "TYPE": "TCC_Negative_Reward",
      "CHECK_INTERVAL":"60000",
      "SOURCE_NODE_ID" : "0.17 0.18"
      "DISTRIBUTION" : "CBR",
      "DATA_SIZE_BYTES" : "50"
    },
    "map": {
      "TIMEOUT": "90",
      "RSSI_RESOLUTION": "20",
      "GRAPH":"GUI"
    }
  }
}
```

O tempo em que a rede simulada é considerada viva corresponde ao período em que todos os nodos fontes ativos possuem ao menos um caminho de roteamento em direção ao sorvedouro da rede, ou seja até o momento em que o algoritmo no controlador pare de enviar novas rotas para a rede. Sendo

assim, para verificar o tempo de vida da RSSF simulada os níveis energéticos recebidos através de *Reports* foram armazenados em um arquivo juntamente com o tempo decorrido desde a inicialização da simulação em segundos.

Com isso, é possível criar um gráfico de nível energético por tempo e avaliar o uso de cada algoritmo verificando por quanto tempo a rede ficou ativa durante a sua execução. Para comparar os desempenhos dos algoritmos se faz necessário normalizar o consumo energético obtido da rede uma vez que o número de nodos utilizados em cada abordagem é diferente. Dessa forma, a Equação 5.2 é aplicada aos níveis energéticos dos nodos a cada segundo na simulação para obter o nível energético da rede ($En_{total}(t)$) em um tempo t quando considerados apenas os módulos utilizados pelo algoritmo.

$$En_{total}(t) = \frac{1}{En_{max} * n} * \sum_{i=1}^{n} x_i(t) \tag{5.2}$$

Onde n é número de nodos utilizados pelo algoritmo, En_{max} é igual a 255 que é o valor máximo energético de um nodo na simulação e $\{x_i\}i \in N$ é uma sequência com os níveis energéticos de todos os nodos utilizados pelo algoritmo em execução no controlador em um dado momento t .

A Tabela 3 mostra a quantidade de nodos utilizados por cada algoritmo e o tempo de vida alcançado por eles, adicionalmente na Figura 29 é ilustrado através de um gráfico o nível energético em razão do tempo decorrido durante a simulação do primeiro cenário para os algoritmos de Djikstra, caminhos disjuntos e recompensa negativa. dessa forma, é possível observar que os algoritmos propostos neste trabalho se destacam em relação ao Djikstra principalmente devido ao fato destes reconfigurarem os caminhos de roteamento sempre que necessário. O que introduz um *overhead* maior na rede, porém é compensado pelo melhor aproveitamento dos nodos disponíveis.

Algoritmo	Quantidade de nodos*	Tempo de vida (s)	En_{total} Final
Djikstra	3	521	0.218
Recompensa Negativa	13	1201	0.160
Caminhos Disjuntos	12	1404	0.037

Tabela 3 – Estatísticas dos algoritmos para o cenário 1. *Quantidade de nodos utilizados pelo algoritmo, exceto pelo fonte e sorvedouro.

Como esperado o algoritmo de Djikstra implementado no controlador original do SDN-WISE tem o pior desempenho com o tempo total de vida de 521 segundos. Isso ocorre pois ele encontra uma única rota no inicio da execução do controlador e não realiza nenhum tipo de manutenção posterior, ou seja não envia nenhuma rota alternativa para substituir a que apresenta falha devido ao esgotamento energético de um ou mais nodos da rede. Isso contribui para o baixo aproveitamento do potencial energético da rede como pode ser constatado pelo fato de que ela morre quando o En_{total} ainda está próxima de 21% do valor máximo em consequência do esgotamento energético de um dos módulos e falta de manutenção dos caminhos de roteamento. Por fim, a reta do gráfico traçada para o Djikstra cai mais acentuadamente que as demais indicando uma sobrecarga sobre os únicos nodos utilizados pelo algoritmo.

Apesar de usar a mesma métrica de número de saltos para encontrar rotas o algoritmo de caminhos disjuntos apresenta uma melhora significativa de 134% em relação ao Djikstra quando considerado o tempo de vida de rede. Isso se deve ao melhor aproveitamento dos nodos disponíveis na rede ao encontrar até três caminhos de roteamento (cenário 1) que são alternados sempre que um dos módulos sendo utilizado tem seu nível energético menor que um limiar mínimo de energia definido no controlador.

Quando comparado com o algoritmo de recompensas negativas o algoritmo de caminhos disjuntos estende a vida da rede em 8,7% tendo portanto o melhor desempenho entre os três algoritmos testados no primeiro cenário. Isso ocorre pois apenas um caminho disjunto é utilizado por vez sendo alternado somente

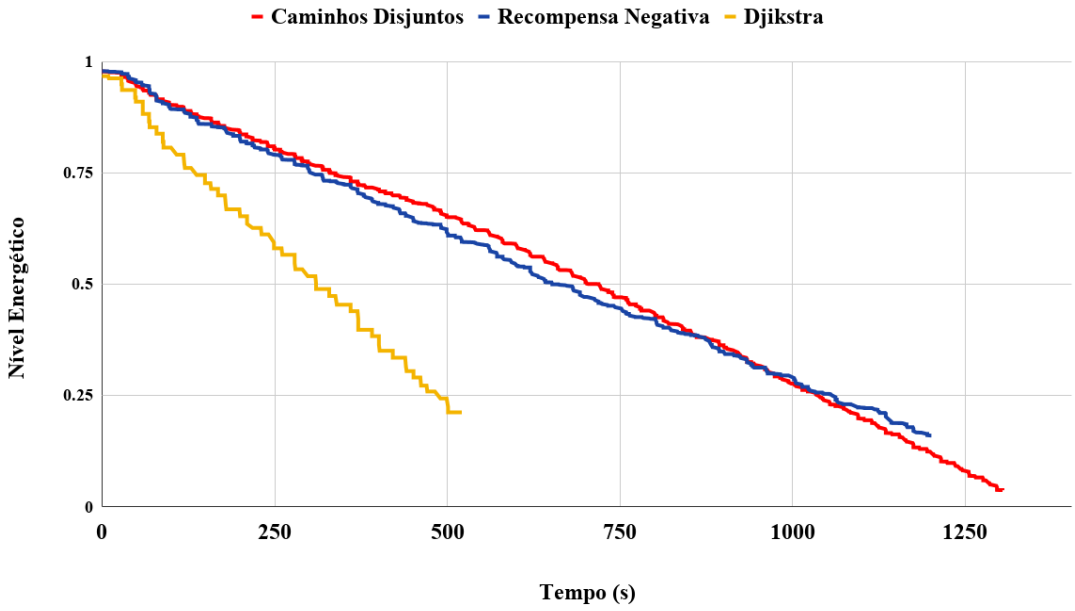


Figura 29 – Cenário 1 - Um nodo fonte e um sorvedouro

quando um dos nodos que o compõem não possa mais transmitir. Em decorrência disso, o restante dos nodos da rede que estão ociosos mudam seu estado para o modo de economia de energia. O que não ocorre com tanta frequência para o caso do algoritmo de recompensas negativas que constantemente alterna os caminhos em uso. Outra consequência deste comportamento mais reativo do algoritmo de recompensas negativas é o aumento de mensagens na rede uma vez que para alterar um caminho se faz necessário enviar duas mensagens (*ClearPath* e *OpenPath*) do controlador para rede.

No segundo cenário o tempo de vida da rede diminui na execução de todos os algoritmos como pode ser visto na Tabela 4, o que é esperado uma vez que existe mais um nodo fonte ativo e portanto o dobro de pacotes de dados gerados além um volume maior de pacotes de configuração sendo enviados pelo controlador. Sendo assim, como visto na Figura 31 o algoritmo de Djikstra apresenta novamente o pior desempenho quando comparado aos demais algoritmos devido aos mesmos motivos explicados para o cenário 1 além de que quando comparado com seu próprio desempenho no primeiro cenário há uma queda de 43,8% no tempo de vida da rede. Isto ocorre pois, como pode ser visto na Figura 30 o caminho mais curto quando considerado o número de saltos é através dos nodos 10, 11 e 12 que portanto serão utilizados pelos dois nodos fontes para escoar seus dados.

Algoritmo	Quantidade de nodos*	Tempo de vida (s)	En_{total} Final
Djikstra	3	293	0.171
Recompensa Negativa	15	800	0.193
Caminhos Disjuntos	7	540	0.041

Tabela 4 – Estatísticas dos algoritmos para o cenário 2. *Quantidade de nodos utilizados pelo algoritmo, exceto pelo fonte e sorvedouro.

O algoritmo de caminhos disjuntos teve um desempenho 60,54% inferior ao seu próprio desempenho no cenário 1 sendo desbancado pelo algoritmo de recompensa negativa. Isso acontece pois diferentemente do ocorrido no primeiro cenário em que o nodo fonte poderia ter até três caminhos, agora um dos nodos fonte terá dois caminhos de roteamento e outro apenas um. Porém, como a rede é considerada morta quando qualquer um dos nodos fontes fica sem um caminho de roteamento o tempo máximo de vida será

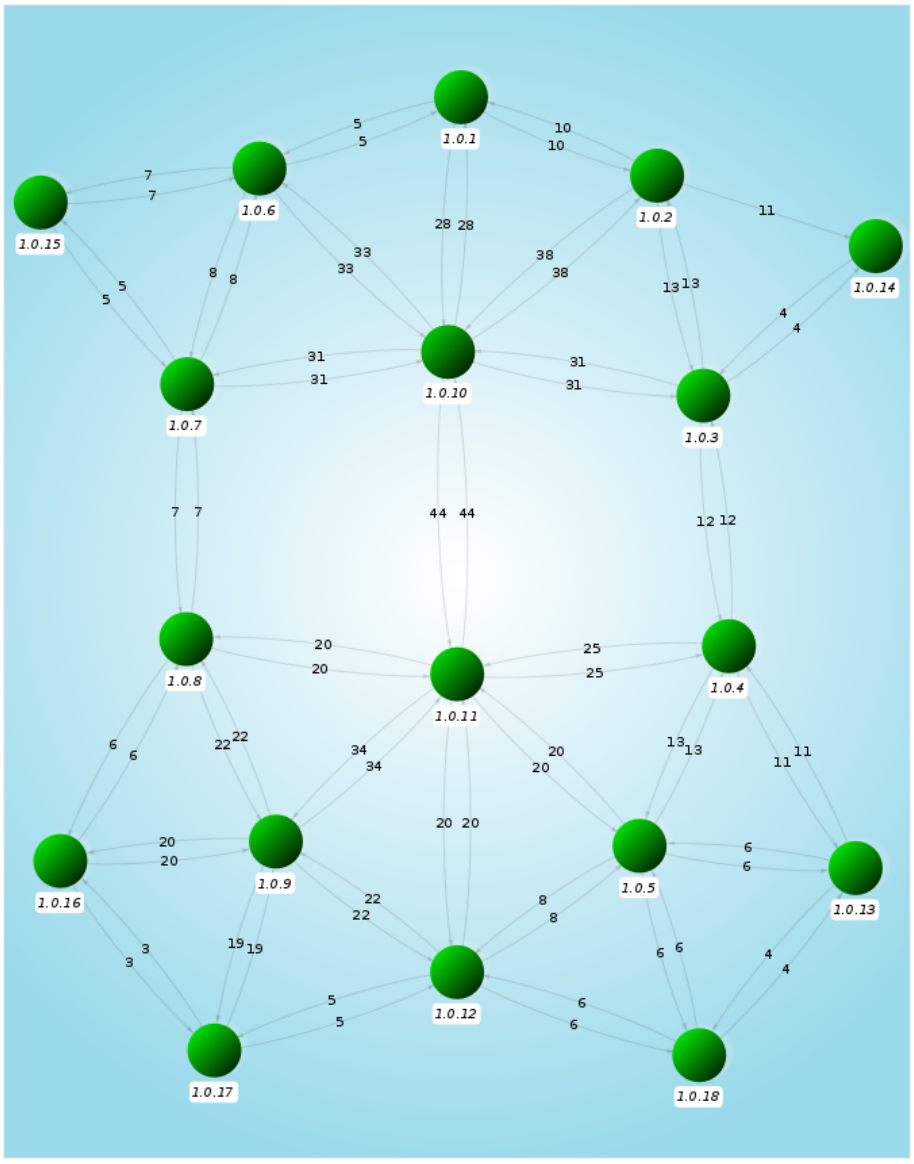


Figura 30 – Topologia da rede vista pelo controlador com pesos dos enlaces calculados de acordo com o algoritmo de recompensas negativas.

o tempo de funcionamento do caminho único atribuído à um dos nodos fontes. Por exemplo, a partir da Figura 30 é possível observar que o nodo fonte 17 pode ter um caminho disjunto através dos nodos 10, 11 e 12 e outro caminho disjunto através do nodo 6 enquanto o nodo fonte 18 teria apenas uma possibilidade de caminho disjunto a através do nodo 2. Dessa forma, o tempo de vida da rede depende diretamente do tempo de vida do caminho ligando o nodo fonte 18 ao sorvedouro através do nodo 2.

O algoritmo de recompensa negativa se destaca no cenário 2 tendo a menor diferença de tempo de vida quando comparado ao cenário 1 (33,39%). Isso ocorre pois neste cenário esse algoritmo utiliza todos os nodos disponíveis e que portanto que melhor balanceia o consumo energético da rede. Portanto, apesar da desvantagem de ser o algoritmo que mais introduz *overhead* na rede devido à manutenção constante dos caminhos, isto é compensado com o roteamento mais eficiente.

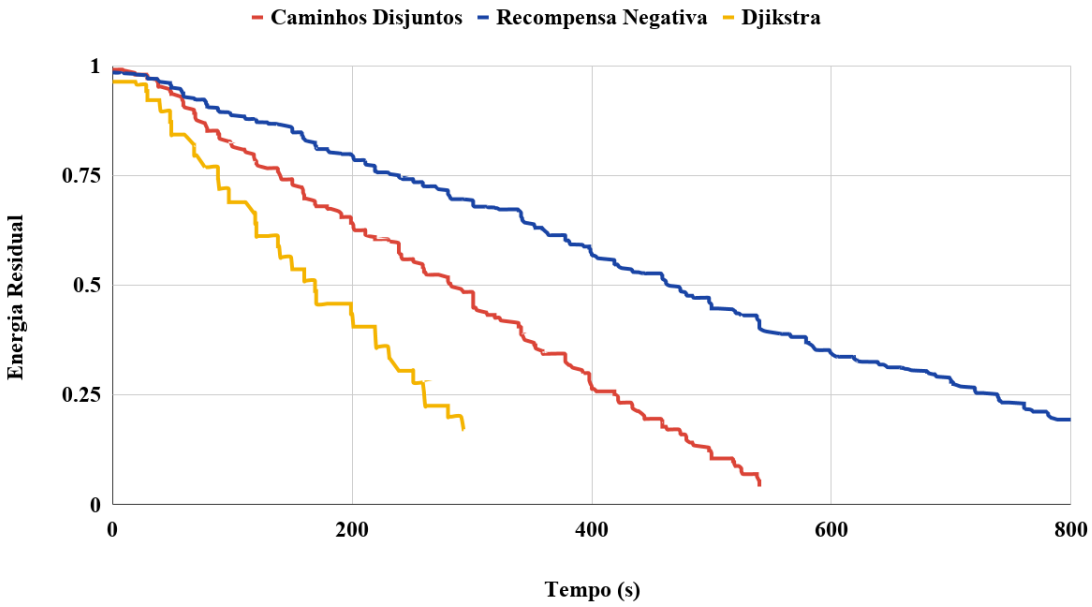


Figura 31 – Cenário 2 - Dois nodos fonte e um sorvedouro

6 CONCLUSÕES

Este trabalho propôs o uso de dois algoritmos de roteamento executados em um controlador SDN para estender o tempo de vida de uma rede de sensores sem fio. Para isso, foram utilizadas a solução SDN-WISE para realizar a comunicação entre camadas de dados e controle do SDN além da ferramenta de simulação Cooja para simular o nodos sensores.

A eficiência energética das RSSF é considerada de suma importância devido às limitações de bateria dos nodos sensores e o custo de manutenção deste tipo de rede. Por isso o roteamento visando estender o tempo de vida é indispensável na maioria das aplicações. Dessa forma, este trabalho explorou a inteligência centralizada de um controlador SDN para executar algoritmos de roteamento que visam estender o tempo de vida da RSSF e enviam caminhos para os nodos de forma dinâmica baseando-se na topologia da rede e em parâmetros fixos definidos em um arquivo de configuração.

O primeiro algoritmo proposto utiliza caminhos disjuntos para rotear os dados dos nodos fontes até o sorvedouro e se mostrou altamente eficiente em um cenário com apenas um nodo fonte gerando dados. Porém, quando mais um nodo foi adicionado à rede o segundo algoritmo de roteamento proposto que utiliza recompensa negativas se mostrou mais eficiente. Entretanto, nos dois cenários de testes criados ambos se foram mais eficientes que o algoritmo de Djikstra utilizado no SDN-WISE quando considerado o tempo de vida da rede.

6.1 Trabalhos Futuros

Como sugestão para trabalhos futuros que podem complementar os resultados obtidos neste trabalho, os seguintes tópicos são citados:

- Utilizar outras métricas como o estado do enlace para realizar roteamento;
- Implementar o uso de mais estados nos módulos sensores simulados afim de explorar a eficiência energético dos nodos;
- Implementar os algoritmos propostos neste trabalho em uma rede de sensores física utilizando o porte do SDN-WISE para o sistema operacional Contiki;

6.2 Repositório de códigos

Os códigos implementados para o controlador SDN e os nodos simulados estão disponíveis em:

- https://github.com/AndreFWeber/TCC

REFERÊNCIAS

AKYILDIZ, I. F.; MELODIA, T.; CHOWDURY, K. R. Wireless multimedia sensor networks: A survey. *Wireless Personal Communications*, v. 14, n. 6, p. 32–39, 2007. Disponível em: <https://ieeexplore.ieee.org/document/4407225/>. Acesso em: 30 de abril de 2018. Citado na página 26.

AL-KARAKI, J. N.; KAMAL, A. E. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, v. 11, n. 6, p. 6 – 28, 2004. Disponível em: <http://ieeexplore.ieee.org/abstract/document/1368893/>. Acesso em: 26 de abril de 2018. Citado na página 30.

ALAZZAWI, L.; ELKATEEB, A. Performance evaluation of the wsn routing protocols scalability. *Journal of Computer Systems, Networks, and Communications*, v. 2008, 2008. Disponível em: <https://www.hindawi.com/journals/jcnc/2008/481046/>. Acesso em: 15 de maio de 2018. Citado na página 30.

ALLIANCE. *What is LoRaWAN^{TM}*. [S.l.], 2015. Disponível em: <https://lora-alliance.org/resource-hub/what-lorawantm>. Citado 4 vezes nas páginas 13, 28, 29 e 30.

ANG, L. minn et al. Chapter 2 wireless multimedia sensor network technology. In: _____. *Wireless Multimedia Sensor Networks on Reconfigurable Hardware*: Information reduction techniques. New Delhi, India: Springer-Verlag Berlin Heidelberg, 2013. p. 5–38. Citado na página 26.

CAPELLA, J. C. et al. A reference model for monitoring iot wsn-based applications. *Sensors*, v. 16, n. 11, p. 1816, 2016. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5134475/>. Acesso em: 26 de abril de 2018. Citado 2 vezes nas páginas 23 e 25.

CENTENARO, M. et al. Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios. *IEEE Wireless Communications*, v. 23, October 2016. Disponível em: <https://arxiv.org/pdf/1510.00620.pdf>. Acesso em: 5 de maio de 2018. Citado na página 28.

DIO, P. D. et al. Exploiting state information to support qos in software-defined wsns. In: IEEE. *Ad Hoc Networking Workshop (Med-Hoc-Net), 2016 Mediterranean*. [S.l.], 2016. p. 1–7. Citado 4 vezes nas páginas 13, 37, 38 e 39.

GALLUCCIO, L. et al. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In: IEEE. *Computer Communications (INFOCOM), 2015 IEEE Conference on*. 2015. p. 513–521. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7218418/>. Acesso em: 24 de maio de 2018. Citado 5 vezes nas páginas 13, 33, 34, 35 e 37.

GUY, C. Wireless sensor networks. *Sixth International Symposium on Instrumentation and Control Technology: Signal Analysis, Measurement Theory, Photo-Electronic Technology, and Artificial Intelligence*, v. 6357, 2006. Disponível em: <https://www.spiedigitallibrary.org/conference-proceedings-of-spie/6357/63571I/Wireless-sensor-networks/10.1117/12.716964.short>. Acesso em: 2 de maio de 2018. Citado na página 25.

HAMID, Z.; HUSSAIN, F. B. Qos in wireless multimedia sensor networks: A layered and cross-layered approach. *Wireless Personal Communications*, v. 75, n. 1, p. 729–757, 2014. Disponível em: <https://link.springer.com/article/10.1007%2Fs11277-013-1389-0>. Acesso em: 30 de abril de 2018. Citado na página 26.

KLAUBERG, A. F. Isolamento das redes virtuais (vlan) em uma infraestrutura em nuvem usando uma abordagem sdn/openflow. Monografia (Tecnólogo em Sistemas de Telecomunicações) - Instituto Federal de Santa Catarina, R. José Lino Kretzer, 608 - Praia Comprida, São José - SC, 88103-310, 2016. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php/Isolamento_das_Redes_Virtuais_(VLAN)_em_uma_Infraestrutura_em_Nuvem_Usando_uma_Abordagem_SDN/OpenFlow>. Acesso em: 18 de junho de 2018. Citado na página 32.

MBARUSHIMANA, C.; SHAHRABI, A. Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks. *Advanced Information Networking and Applications Workshops*, 2007. Disponível em: <https://ieeexplore.ieee.org/document/4224182/>. Acesso em: 30 de abril de 2018. Citado na página 32.

NASIR, H. J. A.; KU-MAHAMUD, K. R. Wireless sensor network: A bibliographical survey. *Indian Journal of Science and Technology*, v. 9, n. 38, 2006. Disponível em: <https://www.spiedigitallibrary.org/conference-proceedings-of-spie>. Acesso em: 5 de maio de 2018. Citado na página 25.

NDIAYE, M.; HANCKE, G. P.; ABU-MAHFOUZ, A. M. Software defined networking for improved wireless sensor network management: A survey. *Sensors*, v. 17, n. 5, 2017. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5469636/>. Acesso em: 20 de maio de 2018. Citado 3 vezes nas páginas 23, 32 e 33.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 3, p. 1617–1634, 2014. Citado 2 vezes nas páginas 32 e 33.

NÚNEZ, G. A.; MARGI, C. B. Energy map model for software-defined wireless sensor networks. p. 826–830, 2017. Disponível em: <http://www.sbrt.org.br/sbrt2017/anais/1570361376.pdf>. Acesso em: 10 de Novembro de 2018. Citado 3 vezes nas páginas 13, 42 e 43.

OLIVEIRA, G. C. de. Localização indoor utilizando a tecnologia lorawan e aprendizado de máquina. Monografia (Bacharel em Engenharia de Telecomunicações) - Instituto Federal de Santa Catarina, R. José Lino Kretzer, 608 - Praia Comprida, São José - SC, 88103-310, 2017. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/images/a/ad/TCC290_Giulio_Cruz_de_Oliveira.pdf>. Acesso em: 12 de maio de 2018. Citado na página 29.

OSTERLIND, F. et al. Cross-level sensor network simulation with cooja. In: IEEE. *Local computer networks, proceedings 2006 31st IEEE conference on*. 2006. p. 641–648. Disponível em: <https://core.ac.uk/download/pdf/11433377.pdf>. Acesso em: 18 de Novembro de 2018. Citado na página 50.

RADI, M. et al. Multipath routing in wireless sensor networks: survey and research challenges. *Sensors*, Molecular Diversity Preservation International, v. 12, n. 1, p. 650–685, 2012. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3279234/>. Acesso em: 18 de junho de 2018. Citado 2 vezes nas páginas 13 e 31.

SARAIVA, L. da S. Plataforma de desenvolvimento de aplicações iot utilizando redes lorawan. Monografia (Bacharel em Engenharia de Telecomunicações) - Instituto Federal de Santa Catarina, R. José Lino Kretzer, 608 - Praia Comprida, São José - SC, 88103-310, 2017. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php/Plataforma_de_desenvolvimento_de_aplica%C3%A7%C3%B5es_IoT_utilizando_redes_LoRaWAN>. Acesso em: 12 de maio de 2018. Citado 2 vezes nas páginas 28 e 29.

SDN-WISE. Sdn-wise. 2017. Disponível em: <http://sdn-wise.dieei.unict.it/>. Acesso em: 24 de maio de 2018. Citado 2 vezes nas páginas 13 e 36.

SDN-WISE. Sdn-wise core. 2017. Disponível em: <http://sdn-wise.dieei.unict.it/docs/guides/Core.html>. Acesso em: 24 de maio de 2018. Citado na página 36.

SEMPREBOM, T. *Explorando descartes de ativações periódicas para provimento de qualidade de serviço em redes IEEE 802.15.4*. Tese (Doutorado) — Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Campus Reitor João David Ferreira Lima, s/n - Trindade, Florianópolis - SC, 88040-900, 2012. Disponível em: <http://repositorio.ufsc.br/xmlui/handle/123456789/100591>. Acesso em: 5 de maio de 2018. Citado 4 vezes nas páginas 13, 26, 27 e 28.

SHAKSHUKI, E.; XING, X.; MALIK, H. An introduction to wireless multimedia sensor networks. *Handbook of Research on Mobile Multimedia*, v. 2, p. 16, 2009. Disponível em: <https://pdfs.semanticscholar.org/4437/0558cadd9c05916947bd65c2d952f0b217a9.pdf>. Acesso em: 26 de abril de 2018. Citado na página 26.

SHEN, H.; BAI, G. Routing in wireless multimedia sensor networks: A survey and challenges ahead. *Journal of Network and Computer Applications*, v. 71, p. 30–49, 2016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804516301102>. Acesso em: 26 de abril de 2018. Citado na página 26.

SILVA, T. H. B. da. Mapeamento de qualidade de serviço em redes ieee 802.15.4. Monografia (Bacharel em Engenharia de Telecomunicações) - Instituto Federal de Santa Catarina, R. José Lino Kretzer, 608 - Praia Comprida, São José - SC, 88103-310, 2017. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php/Mapeamento_de_Qualidade_de_Servi%C3%A7o_em_Redes_IEEE_802.15.4>. Acesso em: 2 de maio de 2018. Citado na página 27.

SINGH, S. K.; SINGH, M. P.; SINGH, D. K. Routing protocols in wireless sensor networks – a survey. *International Journal of Computer Science and Engineering Survey (IJCSES)*, v. 1, n. 2, p. 63–83, 2010. Disponível em: <https://pdfs.semanticscholar.org/f376/57052b3c8dc64052a1df8ae2f1959f438c19.pdf>. Acesso em: 16 de maio de 2018. Citado na página 30.

SMITHA, B.; ANNAPURNA, D. Software defined network for conservation of energy in wireless sensor network. In: IEEE. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. 2017. p. 591–596. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8389505>. Acesso em: 10 de Novembro de 2018. Citado 3 vezes nas páginas 13, 41 e 42.

TOMOVIĆ, S.; RADUSINOVIĆ, I. Extending the lifetime of wireless sensor network with partial sdn deployment. *Telfor Journal*, v. 8, n. 1, p. 8–13, 2016. Disponível em: <https://www.researchgate.net/publication/311987208_Extending_the_lifetime_of_wireless_sensor_network_with_partial_SDN_deployment>. Acesso em: 15 de Novembro de 2018. Citado 2 vezes nas páginas 13 e 44.

WANG, Y. et al. An energy-efficient sdn based sleep scheduling algorithm for wsns. *Journal of Network and Computer Applications*, Elsevier, v. 59, p. 39–45, 2016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804515000910>. Acesso em: 12 de Novembro de 2018. Citado na página 43.

YUAN, Z. et al. A balanced energy consumption sleep scheduling algorithm in wireless sensor networks. In: IEEE. *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. 2011. p. 831–835. Disponível em: <https://ieeexplore.ieee.org/abstract/document/5982654>. Acesso em: 12 de Novembro de 2018. Citado na página 43.