Roteamento para Estender o Tempo de Vida de Redes de Sensores através de SDN e Caminhos Disjuntos

André Felippe Weber*
andrefelippe.weber@gmail.com
Instituto Federal de Santa Catarina
Campus São José
São José, SC, Brasil

Tiago Semprebom
tisemp@ifsc.edu.br
Área de Telecomunicações
Instituto Federal de Santa Catarina
Campus São José
São José, SC, Brasil

Eraldo Silveira e Silva
eraldo@ifsc.edu.br
Área de Telecomunicações
Instituto Federal de Santa Catarina
Campus São José
São José, SC, Brasil

ABSTRACT

The recent advances in the application of Software Defined Networks (SDN) concepts in Wireless Sensor Networks (WSNs) leads the implementation of centralized algorithms. In this work, the SDN concept is considered, together with two routing procedures, with the purpose of disjoint paths generation, aiming to extend WSN's lifetime. Both approaches are based on Dijkstra algorithm for the selection of minimum paths. The first approach allocates distinct paths for various flows towards the same destination. The second one, penalizes computed paths using an adaptive cost function. Simulation results are performed using the Cooja sensor network simulator.

KEYWORDS

Wireless Sensor Networks, Software Defined Networks, Routing Protocols, WSN's lifetime

1 INTRODUÇÃO

A extensão do tempo de vida de uma rede de sensor sem fio (RSSF) é uma questão primordial que precede outros requisitos comportamentais destas redes. Uma das formas de abordar este problema é por meio do uso de estratégias de roteamento que considerem aspectos de consumo energético nas decisões de encaminhamento de mensagens [1].

Entretanto, o roteamento levando em conta decisões locais de nodos pode impactar o processamento e consumo energético dos nodos de uma RSSF. Os recentes avanços na aplicação de conceitos de redes definidas por software (SDN) no contexto de redes sensores podem vir a favorecer a implantação de algoritmos centralizados ou parcialmente centralizados [2–4].

Em uma SDN, os planos de controle e dados são separados. O primeiro é responsável pelas decisões de encaminhamento e o segundo pelo encaminhamento dos dados. O plano de controle pode estar em localização física distinta do plano de dados e é constituído por um controlador que possui uma visão geral da rede e capacidade para centralizar toda a inteligência, isto é, gerenciar o roteamento, topologia, segurança, QoS e controlar o consumo energético da rede [5]. Em uma RSSF, o controlador pode estar fisicamente posicionado em um *gateway* que a conecta a um sistema com infraestrutura.

Este trabalho se utiliza da plataforma SDN-WISE [4] para implementar duas abordagens de roteamento que objetivam estender o tempo de vida de uma RSSF por meio do conceito de SDN. Ambas favorecem a implantação de caminhos disjuntos e são baseadas na sucessiva aplicação do algoritmo de Dijkstra para computação e posterior seleção de caminhos mínimos. Uma das técnicas propostas, aloca caminhos distintos para diversos fluxos em direção a um mesmo destino. Uma segunda versão penaliza caminhos já computados usando uma função de custo que pode ser adaptada a diferentes condições. A avaliação das abordagens é realizada por meio do simulador de redes de sensores Cooja do Sistema Contiki [6].

O restante deste trabalho é organizado como segue. Na Seção 2 apresentam-se trabalhos relacionados da literatura. A Seção 3 apresenta o sistema proposto e discute os algoritmos utilizados. Na Seção 4 apresenta-se a implementação e avaliação do sistema. Finalmente, na Seção 5 as conclusões e trabalhos futuros são apresentados.

2 TRABALHOS RELACIONADOS

O suporte a SDN para RSSF usado neste trabalho é o SDN-WISE proposto por Galluccio et al. [4]. Trata-se de uma arquitetura baseada no *OpenFlow* que permite criar e comunicar nodos sorvedouros (*Sink*) e fontes (*Source*) com um controlador remoto. O SDN-WISE, diferentemente do *OpenFlow*, implementa funcionalidades que melhoram a eficiência energética da rede. Por exemplo, através dele é possível desligar o rádio dos nodos periodicamente (*duty-cycle*). Ele permite também que os nodos agreguem suas mensagens a pacotes oriundos dos vizinhos, que são roteados através deles. Ademais, o protocolo SDN-WISE prevê que os nodos possuam capacidade de decisão em alguns casos (noção de estados do nodo). Dessa forma, diminui-se a quantidade de interações entre os nodos e o controlador melhorando a eficiência energética da rede.

No que tange ao tempo de vida de redes, existe uma série de trabalhos em parte relatados em Yetgin et al. [1]. Neste *survey*, os autores classificam as diversas abordagens de ampliação da vida das redes, incluindo técnicas baseadas no escalonamento (*duty cycle*), camada de enlace, direcionamento de comunicação (*beamforming*) e roteamento, entre outras. As SDNs não são explicitamente referenciadas.

No trabalho realizado por Smitha and Annapurna [7], as camadas de controle e dados são claramente separadas com o intuito de conservar energia dos nodos sensores (o que é inerente ao SDN). A rede é organizada em agrupamentos. O controlador escolhe um líder de agrupamento (*cluster*), considerando a capacidade energética do nodo e a distância média dele em relação ao restante dos nodos. Os nodos do agrupamento deverão destinar os dados monitorados a um líder. Desta forma, os participantes de cada agrupamento, exceto

o líder, ficam responsáveis apenas por encaminhar dados com base na sua tabela de roteamento.

No trabalho realizado por Núñez and Margi [8] é proposta a criação de um mapa energético para SDWSN. Para isso, o controlador implementa um modelo de previsão do consumo energético dos nodos da rede. Seguindo a arquitetura SDN, um controlador centralizado obtém informações do comportamento da rede e estima a taxa de consumo energético de cada nodo. 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 nodos. Na modelagem do mapa energético foram considerados quatro modos possíveis de operação para os nodos: Transmitindo, Escutando, Processando e Economia de Energia. Em seguida, cada modo é considerado como um estado de uma cadeia de Markov, proporcionando um modelo de predição do comportamento da rede.

No trabalho realizado por Wang et al. [9], 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 nodos da rede. O princípio do mecanismo do tipo *Sleep-Scheduling* é reduzir o consumo energético de determinados nodos, ou seja, reduzir o número de mensagens trocadas, enquanto mantém outros nodos 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 nodos vivos e diminuir a quantidade de nodos isolados quando comparado ao algoritmo EC-CKN.

No trabalho realizado em [10], os autores consideram uma implementação mista em que nodos chamados de (SDNSN), habilitados a integrar uma RSSF baseada no conceito de SDN, coexistem com nodos 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 nodos podem ser controlados pelo controlador SDN. Este trabalho propõe um algoritmo de roteamento que se assemelha ao segundo algoritmo implementado neste artigo, mas existem diferenças. Para cada nodo da SDNSN, é encontrado o caminho mais curto, aplicando Djikstra e usando uma função de custos nos enlaces da forma $C(i,j) = EC(i,j)^{\alpha}/R(i)^{\beta}$. A função EC não é claramente colocada, mas representa a energia usada para transmitir no enlace (i,j) e $R(i)$ a energia residual em i. Os parâmetros α e β podem ser ajustados conforme o cenário.

3 MODELO DO SISTEMA E ESTRATÉGIAS DE ROTEAMENTO IMPLEMENTADAS

Assume-se que os nodos sensores são aleatoriamente distribuídos em um *playgroud*, sendo que um conjunto de N nodos fonte transmite mensagens periodicamente para um único nodo sorvedouro. Os nodos sensores iniciam seu tempo de vida com carga máxima de bateria, sendo que, no caso específico do nodo sorvedouro, não existe restrição energética[1]. Os nodos fonte alcançam o sorvedouro com um ou mais saltos.

[1]Considera-se que o nodo sorvedouro está conectado a uma fonte permanente de energia.

A Fig.1 ilustra o sistema proposto neste trabalho, onde as camadas de controle e dados se comunicam para que novas regras de roteamento sejam criadas e aplicadas aos nodos sensores. Na camada de dados, os nodos implementam o protocolo do SDN-WISE [4] e podem ser utilizados para geração e encaminhamento de dados para o controlador por meio de um sorvedouro da rede. Para isso, eles transmitem informações como nível energético e estado dos enlaces para o controlador que, por sua vez, informa os caminhos de roteamento.

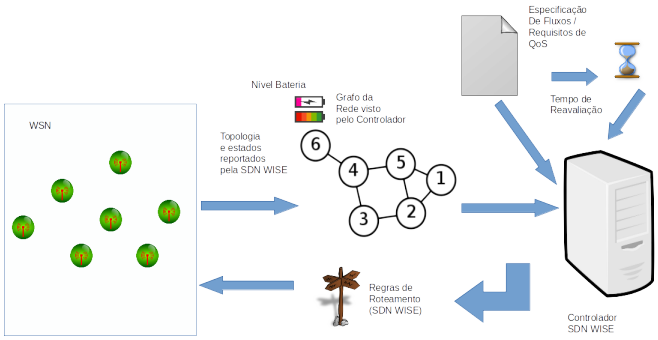

Figura 1: Visão geral do sistema proposto.

Na camada de controle, o controlador cria um grafo com a topologia da rede por meio de informações recebidas dos nodos na camada de dados e o utiliza para encontrar os melhores caminhos de roteamento, de acordo com o algoritmo em uso.

O controlador acessa uma especificação de QoS dos fluxos gerados na rede. Futuramente, pretende-se usar esta especificação para tomada de decisão de roteamento. Por ora, ela permite informar limiares de energia e informação de temporização para disparo de uma nova computação de rota por parte do controlador.

São propostos dois algoritmos no controlador que utilizam abordagens distintas com o propósito de estender o tempo de vida da rede. Estes algoritmos são descritos a seguir.

3.1 Caminhos Completamente Disjuntos

O objetivo de criar caminhos disjuntos é dividir os diversos fluxos de dados gerados pelos nodos fontes por rotas que possuem nodos exclusivos [11, 12]. A aplicabilidade deste tipo de algoritmo é evidenciada, principalmente, em redes com alta densidade de nodos. Esta abordagem contribui com o aumento do tempo de vida dos nodos, uma vez que o consumo de energia é limitado à carga gerada em um único fluxo de dados. Ademais, o algoritmo proposto mantém o caminho ativo, se necessário, até que o nível energético de um dos nodos que o compõem ultrapasse um limiar mínimo energético aceitável. Como consequência, o restante dos nodos da rede que estão ociosos podem entrar em modo de economia de energia.

Este algoritmo utiliza caminhos disjuntos para encontrar rotas entre os nodos fontes e o sorvedouro de uma RSSF. Assim sendo, todos os caminhos possíveis e disjuntos entre os dois nodos de interesse são encontrados utilizando como métrica o menor número de saltos, e, em seguida, verifica-se se todos os nodos possuem nível

Tabela 1: Exemplo: Níveis energéticos por nodo.

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

energético acima de um limiar mínimo exigido e finalmente escolhe-se o caminho que possua o maior somatório das energias residuais de seus nodos. Por exemplo, um controlador que foi configurado com um limiar mínimo igual a 5, encontra dois caminhos, A e B, compostos pelos nodos [1, 2, 3, 6] e [1, 4, 5, 6], respectivamente. Todos os nodos possuem seus níveis energéticos mostrados na Tabela 1, sendo que o nodo 1 é o sorvedouro e o nodo 6, o fonte. Assim sendo, o caminho escolhido como melhor será B, com somatório igual a 710, pois, apesar de A possuir o maior somatório dos níveis energéticos de seus nodos (763), o nodo 3 possui nível energético abaixo do limiar mínimo.

Algoritmo 1: *DISJOINT PATH* (Etapa 1)

```
   Entrada: NetGraph, PathRequestPacket
   Saída: Caminho de roteamento
1  início
2      enquanto true faça
3          Djikstra.init(NetGraph)
4          nodeAddr = PathRequestPacket.dst;
5          sinkAddr = PathRequestPacket.src;
6          Path ← Djikstra.getPath(sinkAddr, nodeAddr)
7          se Path encontrado então
8              PathsFound.push(Path);
9              para cada node ∈ Path faça
10                 se node.address != sinkAddr e node.address !=
                       nodeAddr então
11                     Remove nodo do grafo
12                     NetGraph.remove(node);
13                 senão
14                     Mantém nodo no Grafo
15                 fim
16             fim
17         senão
18             retorna PathChecker(PathsFound);
19         fim
20     fim
21 fim
```

O pseudocódigo do algoritmo de caminhos disjuntos pode ser separado em duas etapas lógicas. O Algoritmo 1 mostra a primeira etapa do processo o qual objetiva encontrar todos os possíveis caminhos disjuntos e que são executados quando uma requisição de caminho é recebida da camada de dados. O Algoritmo 2, por sua vez, apresenta a segunda etapa (*PathChecker*) que envolve o processo de escolha do melhor caminho, considerando o nível energético dos nodos. Esta última etapa é executada pelo controlador SDN a cada intervalo de tempo definido em seu arquivo de configuração para verificar se o caminho ativo ainda é considerado o melhor.

O Algoritmo 1 recebe como entrada um pacote de requisição de caminho oriundo do sorvedouro da RSSF e um grafo com a topologia da rede (*NetGraph*) que é utilizado para inicializar um objeto do tipo Djikstra. Na linha 6 do Algoritmo 1, o objeto Djikstra encontra o melhor caminho (com menor número de saltos). Em seguida (linha 8), o caminho é armazenado e, na linha 12, os nodos do caminho encontrado são removidos do grafo da rede pois, com exceção dos nodos fonte e sorvedouro, estes não poderão mais ser utilizados em outro caminho. Por fim, na linha 18, o algoritmo deve retornar o resultado do *PathChecker*.

Algoritmo 2: *PATHCHECKER* (Etapa 2)

```
   Entrada: PathsFound
   Saída: bestPath
1  início
2      para cada Path ∈ PathsFound faça
3          para cada node ∈ Path faça
4              se node.bateria > limiar && ! node.active 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
```

A segunda etapa (Algoritmo 2), recebe como entrada uma lista com todos os caminhos encontrados para cada nodo fonte da rede no Algoritmo 1. O Algoritmo 2 pode ser executado uma vez para encontrar um novo caminho e periodicamente para verificar se todos os nodos do caminho ativo ainda possuem seu nível energético acima do limiar mínimo definido no controlador. Portanto, na linha 4 do Algoritmo 2, o nível energético da bateria dos nodos 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 (linha 8). Adicionalmente, é verificado que o nodo em questão não está ativo em nenhum outro caminho. Em seguida, com todos os caminhos verificados somente aquele com menor número de saltos será retornado (*bestPath*). Por fim, é válido observar que esta abordagem mantêm a escolha de um caminho até que um dos nodos ativos tenha seu nível de energia reduzido abaixo do limiar mínimo.

3.2 Recompensas Negativas

A segunda abordagem utiliza o nível energético residual dos nodos como métrica para encontrar os caminhos, além de aplicar recompensas negativas (*Negative Reward*) aos enlaces em uso pela rede.

Esta abordagem também busca favorecer caminhos disjuntos aumentando o custo de caminhos escolhidos a cada execução de um algoritmos de mínimo custo.

Esta segunda abordagem de roteamento prevê o balanceamento do consumo energético da rede ao penalizar os enlaces que estão sendo utilizados em um caminho ativo já selecionado. A ideia é não remover um caminho do grafo após este ser selecionado, mas desencorajar a escolha de um caminho já escolhido aplicando recompensas negativas ao dobrar o custo de uso dos enlaces dos nodos 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 nodos possuam melhor nível energético residual.

A vantagem deste algoritmo em relação ao anterior é 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 nodos seja igual ou próximo a zero. Além disto, quando a rede não é muito densa, existe reaproveitamento de partes de caminhos já usados por determinados fluxos, mas que ainda possuem limiar energético aceitável.

Algoritmo 3: Recompensas Negativas

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

O algoritmo de recompensa negativa (Algoritmo 3) recebe como entrada um pacote de requisição de caminho oriundo do sorvedouro da RSSF, um grafo (*NetGraph*) da rede e uma lista (*EnergyMap*) com o nível energético de todos os nodos da rede. Na linha 3 do Algoritmo 3, a função $setWeight()$ calcula os custos iniciais a serem associados a cada arco. Em uma primeira versão, construiu-se uma função baseada em uma visão experimental que poderá ser aperfeiçoada futuramente. O peso de cada aresta do grafo é calculado com o valor máximo de energia residual (Max_{Enr}) de cada nodo, multiplicado por dois, menos o somatório da energia residual (*Enr*) atual dos nodos em cada extremidade de uma aresta (Equação 1). A subtração é feita para que os enlaces com maior nível energético residual possuam o menor peso possível. 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 nodos descritos na Tabela 1 tem seus enlaces representados como E[M1, M2], onde E é o enlace, M1 e M2 representam os nodos e o Max_{Enr} de um nodo é 255. Sendo assim, se a rede 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 1 serão 5, 320 e 1240 para os enlaces A, B e C, respectivamente.

$$weight = (2 * Max_{Enr} - Enr0 + Enr1) * (2^n) \quad (1)$$

Ao encontrar um caminho, os pesos atribuídos aos enlaces que o compõem são dobrados (linha 10), fazendo com que os caminhos encontrados nas demais iterações do algoritmo distribuam o consumo energético na rede entre os diversos nodos que a compõem. Por fim, todos os caminhos encontrados para o nodo fonte ($PathRequestPacket.dst$) são retornados na lista *PathsFound*.

Assim, como apresentado nos Algoritmos 1 e 2 para os caminhos disjuntos, o controlador verifica se os caminhos escolhidos possuem todos os seus nodos com nível energético acima de um limiar mínimo definido para a rede e, periodicamente, executa o Algoritmo 3 para assegurar que os caminhos escolhidos permanecem como a melhor escolha, de acordo com a abordagem de recompensas negativas.

4 AVALIAÇÃO DA PROPOSTA

Esta seção apresenta as principais ferramentas, os aspectos de implementação e os cenários utilizados para avaliar as propostas discutidas nesse trabalho.

4.1 Ferramentas Usadas

O simulador utilizado para avaliar a proposta foi o Cooja [13]. Trata-se de um simulador de redes de sensores sem fios capaz de executar programas escritos para o sistema operacional Contiki, executando diretamente na CPU do *host* ou, em nível de instrução, em conjunto com o MPSIM que emula instruções do microcontrolador TI MSP430.

O suporte a SDN utilizado foi o SDN-WISE [4]. Os criadores deste sistema disponibilizaram os códigos dos nodos fonte e sorvedouro, bem como exemplos de controladores para a comunidade acadêmica. Os nodos usados foram escritos em Java, já que se trata de alternativa a emulação de nodos físicos do Contiki/Cooja.

No SDN-WISE, 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, a receber regras de roteamento, além de pacotes de configuração e dados. Em um segundo momento, o sorvedouro utiliza o mecanismo de descoberta da topologia para que os nodos da rede encontrem o vizinho que represente o melhor salto em direção ao sorvedouro. Será por intermédio desse vizinho que os pacotes de *Report* serão enviados para o controlador.

Após verificar a disponibilidade e se conectar ao controlador, o sorvedouro inicia o processo de descoberta da topologia da rede. Para isso, ele envia por meio de difusão, um *Beacon* que possui apenas *bytes* de *payload* anexados aos dez *bytes* do cabeçalho padrão do SDN-WISE. Ao encontrar o próximo salto para chegar até o sorvedouro mais próximo, os nodos iniciam o envio dos pacotes de

Report para o controlador. É através desta mensagem que é possível indicar o estado dos enlaces, tais como RSSI e nível de bateria dos nodos.

O controlador é executado como um servidor e portanto na inicialização do seu processo uma conexão TCP é aberta para que um sorvedouro se conecte no controlador. O projeto do controlador disponibilizado pelos autores 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.

O SDN-WISE utiliza o GraphStream[2] - uma biblioteca pública para projetos em Java que permite modelar e analisar grafos dinamicamente. Ela é utilizada pelo simulador para plotar a topologia da rede simulada.

4.2 Aspectos da Implementação do Coordenador

Neste trabalho criou-se um novo controlador que pode executar os algoritmos de caminhos disjuntos, recompensas negativas e Djikstra (este foi reproduzido a partir do controlador SDN-WISE). Ademais, o controlador 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, a leitura de um arquivo de configuração permite obter o endereço e IP porta, além de outros parâmetros utilizados na conexão entre o controlador e a camada de dados. Na seção *map* do arquivo, é possível definir (i) o tempo máximo que um nodo pode ficar sem enviar *Report* para o controlador antes de ser excluído do grafo da rede, (ii) um valor para a máxima variação de RSSI admitida e (iii) 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 nodos da rede. Alguns campos adicionais são utilizados para configurar os nodos fonte da rede, o perfil de tráfego gerado e tamanho de pacotes.

Com o grafo criado e o arquivo de configuração lido, o controlador deve configurar os nodos fontes da rede, o que é realizado a cada *Report* recebido.

Após a execução do algoritmo e envio de rotas, o controlador inicia um temporizador para que seja periodicamente verificado se ainda existem caminhos disponíveis entre os nodos fonte e o sorvedouro. Caso exista, é verificado se os caminhos em uso continuam sendo os melhores, de acordo com o algoritmo em uso, ou se devem ser alterados. Caso contrário, a rede é considerada morta.

4.3 Cenários Simulados

Dois cenários distintos foram investigados. Em ambos os cenário os nodos da rede foram aleatoriamente distribuídos No **Cenário 1** o número de nodos foi variado de 25 a 125 nodos, sendo o número de nodos fonte fixado em 5. O objetivo é aumentar a densidade da rede e proporcionar caminhos alternativos para os algoritmos. No **Cenário 2**, o total de nodos foi fixado em 100, variando-se os nodos fontes de 5 a 25 com passo de 5. Neste caso, objetiva-se comparar os níveis de energia da rede com os algoritmos propostos.

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

Nos cenários simulados, os nodos foram distribuídos de tal forma que todos possuam ao menos um vizinho em seu raio de cobertura. O meio de transmissão escolhido no Cooja foi o UDGM (*Unit Disk Graph Model*) que permite selecionar 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, 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 nodo possuem o mesmo RSSI (255), ou seja, a distância física entre dois nodos não é considerada nas simulações. Outrossim, o limiar energético mínimo para os nodos encaminharem pacotes na rede foi definido como 5. A periodicidade de transmissão de dados dos nodos fonte para o sorvedouro é de 2 segundos. Por fim, o sorvedouro está diretamente conectado com o controlador SDN.

O tempo em que a rede simulada é considerada viva (tempo de vida) 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 controlador pare de enviar novas rotas para a rede.

4.4 Análise de Resultados

A Fig.2 apresenta o tempo de vida para as abordagens avaliadas Cenário 1. É possível observar que os algoritmos propostos neste trabalho se destacam em relação a pura aplicação do algoritmo de Djikstra, principalmente devido ao fato destes reconfigurarem os caminhos de roteamento sempre que necessário. O que introduz *overhead* maior na rede, que, porém é compensado pelo melhor aproveitamento dos nodos disponíveis. A Tabela 2 apresenta o número médio de mensagens de controle trocadas em cada uma das abordagens avaliadas no Cenário 1.

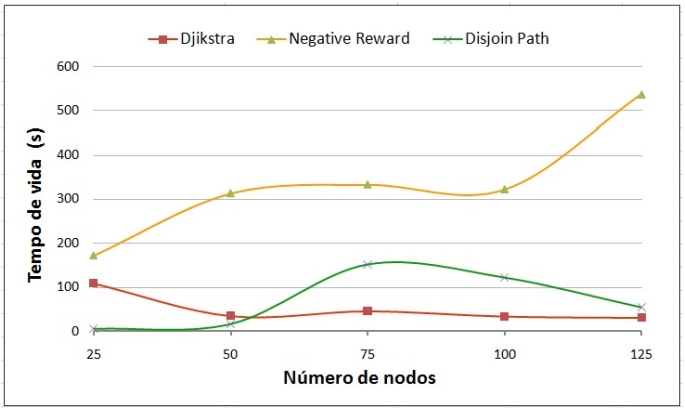

Figura 2: Tempo de vida variando o número de nodos.

Como esperado, o algoritmo de Djikstra mantêm o tempo de vida relativamente pequeno e constante. Isso ocorre pois ele encontra uma única rota no início 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.

Tabela 2: Número médio de mensagens de controle cenário 1.

Abordagem	número de mensagens de controle
Djikstra	692
Caminhos Disjuntos	1650
Recompensa Negativa	6241

Apesar de usar a mesma métrica de número de saltos para encontrar rotas, o algoritmo de caminhos disjuntos apresenta tempo de vida maior quando existem caminhos alternativos. Isso se deve ao melhor aproveitamento dos nodos disponíveis na rede, sendo que novos caminhos (disjuntos) são ativados quando nodos tem seu nível energético menor que o limiar mínimo de energia definido no controlador.

Observa-se que o tempo de vida do algoritmo de recompensa negativa é superior aos demais. Isso ocorre pois, no algoritmo de caminhos disjuntos, apenas um caminho é utilizado por vez, sendo alternado somente quando um dos nodos que o compõem não possa mais transmitir mensagens. 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.

A Fig.3 mostra o resultado da energia normalizada para o Cenário 2. Aqui, novamente, fica evidenciado o melhor desempenho do algoritmo de recompensa negativa. O número médio de mensagens de controle trocadas nas abordagens avaliadas no Cenário 2 podem ser observadas na Tabela 3.

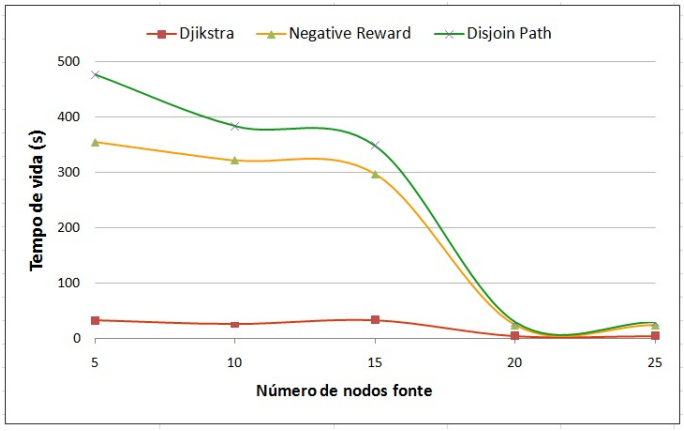

Figura 3: Tempo de Vida para o Cenário 2.

Tabela 3: Número médio de mensagens de controle do cenário 2.

Abordagem	número de mensagens de controle
Djikstra	402
Caminhos Disjuntos	1182
Recompensa Negativa	4475

Nota-se que algoritmo de recompensa negativa aumenta o número de mensagens na rede uma vez que para alterar um caminho se faz necessário enviar duas mensagens (*ClearPath* e *OpenPath*) do controlador para rede. Apesar do *overhead* introduzido pelo algoritmo, devido a manutenção constante dos caminhos, há a compensação com o roteamento mais eficiente. As Tabelas 2 e 3 mostram a quantidade de mensagens de controle trocadas

5 CONCLUSÕES

Neste artigo foi apresentada uma proposta de uso de redes definidas por software no contexto de redes de sensores, com o objetivo de estender o tempo de vida da rede.

Foram investigados dois algoritmos que usam a abordagem de caminhos disjuntos tendo sido comparados com o algoritmo de caminhos mínimos de Djikstra. Os resultados apresentados mostraram-se promissores e demonstram a factibilidade da abordagem de SDN para esta finalidade. Ficou evidenciado, porém, que existe um *overhead* que pode ser significativo dependendo da periodicidade das mensagens trocadas pelo suporte da SDN. O ajuste fino destes parâmetros temporais pode reduzir este efeito.

Pretende-se em trabalhos futuros aprimorar as funções de custo que foram usadas, de forma a considerar outros estimadores de estado de enlace e também informações advindas da camada de enlace. A comparação em termos de desempenho com protocolos distribuídos também deve ser objeto de pesquisa.

Como o suporte SDN-WISE usa o conceito de estados para intervenções locais no processo de encaminhamento, pode-se também vislumbrar o uso destes mecanismos para obter um protocolo parcialmente centralizado.

INFORMAÇÕES ADICIONAIS

Este artigo é resultado do trabalho de conclusão de curso do autor André Fellipe Weber, que por sua vez foi em parte fundamentado nas discussões geradas no contexto do Projeto Interno de Pesquisa de título: "Investigação sobre Técnicas de QoS em Redes de Sensores e IoT".

REFERÊNCIAS

[1] Halil Yetgin, Kent Tsz Kan Cheung, Mohammed El-Hajjar, and Lajos Hanzo Hanzo. A survey of network lifetime maximization techniques in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 19(2):828–854, 2017. URL https://ieeexplore.ieee.org/document/7812629.

[2] Bruno Trevizan de Oliveira, Cintia Margi, and Lucas Batista Gabriel. Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks. *2014 IEEE Latin-America Conference on Communications, IEEE LATINCOM 2014*, 13, 02 2015. doi: 10.1109/LATINCOM.2014.7041885.

[3] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[4] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 513–521. IEEE, 2015. URL https://ieeexplore.ieee.org/abstract/document/7218418/.

[5] Musa Ndiaye, Gerhard P. Hancke, and Adnan M. Abu-Mahfouz. Software defined networking for improved wireless sensor network management: A survey. *Sensors*, 17(5), 2017. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5469636/.

[6] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Nov 2004. doi: 10.1109/LCN.2004.38.

[7] B Smitha and D Annapurna. Software defined network for conservation of energy in wireless sensor network. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 591–596. IEEE, 2017. URL https://ieeexplore.ieee.org/abstract/document/8389505.

[8] Gustavo A Núñez and Cintia B Margi. Energy map model for software-defined wireless sensor networks. pages 826–830, 2017. URL http://www.sbrt.org.br/sbrt2017/anais/1570361376.pdf.

[9] Yanwen Wang, Hainan Chen, Xiaoling Wu, and Lei Shu. An energy-efficient sdn based sleep scheduling algorithm for wsns. *Journal of Network and Computer Applications*, 59:39–45, 2016. URL https://www.sciencedirect.com/science/article/pii/S1084804515000910.

[10] Slavica Tomović and Igor Radusinović. Extending the lifetime of wireless sensor network with partial sdn deployment. *Telfor Journal*, 8(1):8–13, 2016. URL https://www.researchgate.net/publication/311987208_Extending_the_lifetime_of_wireless_sensor_network_with_partial_SDN_deployment.

[11] Bharat Bhushan and G Sahoo. A comprehensive survey of secure and energy efficient routing protocols and data collection approaches in wireless sensor networks. In *Signal Processing and Communication (ICSPC), 2017 International Conference on*, pages 294–299. IEEE, 2017. URL https://ieeexplore.ieee.org/abstract/document/8305856.

[12] Marjan Radi, Behnam Dezfouli, Kamalrulnizam Abu Bakar, and Malrey Lee. Multipath routing in wireless sensor networks: survey and research challenges. *Sensors*, 12(1):650–685, 2012. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3279234/.

[13] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local computer networks, proceedings 2006 31st IEEE conference on*, pages 641–648. IEEE, 2006. URL https://core.ac.uk/download/pdf/11433377.pdf.