Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

ESTUDO E IMPLEMENTAÇÃO DE ALGORITMOS DE ROTEAMENTO PARA REDE DE SENSORES SEM FIO UTILIZANDO EFICIÊNCIA ENERGÉTICA COMO MÉTRICA

RONALDO TAFAREL PEREIRA DE SOUZA

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS/MS

2019

ESTUDO E IMPLEMENTAÇÃO DE ALGORITMOS DE ROTEAMENTO PARA REDE DE SENSORES SEM FIO UTILIZANDO EFICIÊNCIA ENERGÉTICA COMO MÉTRICA

RONALDO TAFAREL PEREIRA DE SOUZA

Este exemplar corresponde ao trabalho de conclusão de curso da disciplina Projeto Final de Curso devidamente corrigida e defendida por Ronaldo Tafarel Pereira de Souza e aprovada pela banca examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS/MS
2019

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

ESTUDO E IMPLEMENTAÇÃO DE ALGORITMOS DE ROTEAMENTO PARA REDE DE SENSORES SEM FIO UTILIZANDO EFICIÊNCIA ENERGÉTICA COMO MÉTRICA

RONALDO TAFAREL PEREIRA DE SOUZA

Novembro de 2019

Banca Examinadora:

Prof. Dr. Rubens Barbosa Filho (Orientador)
Área de Computação – UEMS

Prof. MSc. André Chastel Lima
Área de Computação – UEMS

Profa. Dr. Osvaldo Vargas Jaques
Área de Computação – UEMS

Esse trabalho é dedicado aos meus pais, que me apoiaram na decisão de sair da Bahia e vir para o Mato Grosso do Sul me tornar um profissional.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelo dom da vida, e pela força nos momentos difíceis, não somente durante essa caminhada, mas durante a vida.

Agradeço também ao meu orientador, que me auxiliou durante esse trabalho, me proporcionando experiências que levarei para vida.

Agradeço a Professora Dra. Gláucia Gabriel Sass e ao Prof Dr. Odival Faccenda, por me proporcionar a primeira iniciação científica.S

Agradeço aos meus amigos que me apoiaram nessa etapa da minha vida, bem como a minha namorada, que esteve comigo durante quase toda a minha graduação.

RESUMO

Os sensores que participam das redes de sensores sem fio, são dispositivos limitados, tanto que sozinhos não são capazes de realizar grandes tarefas, porém em grandes quantidades esses dispositivos podem realizar tarefas mais complexas. As limitações desses sensores estão ligadas à componentes de processamento, memória e capacidade energética. Por ser o componente que permite o funcionamento da rede por tempo limitado, a capacidade energética desses dispositivos merece atenção. Diante disso, esse estudo visa estudar e implementar dois algoritmos de roteamento baseado no algoritmo de Dijkstra utilizando como métrica a eficiência energética. Sendo assim, foi realizado um estudo de campo, e desenvolvido ambos os métodos utilizando a linguagem C, *opengl,* e uma API do *gnuplot*. Após a coleta de dados, os mesmos foram transformados em gráficos, sendo possível observar a melhor utilização da rede por parte do método de caminhos múltiplos em relação ao método de caminho único.

Palavras-chave: Sensores sem fio. Método de roteamento. Eficiência energética.

SUMÁRIO

1. INTRODUÇÃO…………………………..…………………………………….. 17
2. REFERENCIAL TEÓRICO……………………..…………………………….. 21
2.1. TRABALHOS SEMELHANTES………………………………………………… 21
2.2. FERRAMENTAS………………………………………………………………. 23
2.2.1. LINGUAGEM C……………….………………………………………… 23
2.2.2. OPENGL………………………………………………………………….24
2.2.3. GNUPLOT ……………………..…………………………………………… 24
3. METODOLOGIA……………………………..…………………………………..... 27
4. DESENVOLVIMENTO……….…………….………………………………….….29
5. ANÁLISE E INTERPRETAÇÃO DOS RESULTADO…………………..………..… 43
5.1. TELAS ……………………………………………………………………….43
5.2 REPRESENTAÇÃO GRÁFICA DOS RESULTADOS …………………….45
6. CONCLUSÃO ………………………………………………………………..…..57
REFERÊNCIAS BIBLIOGRÁFICAS……………………………..……………….. 59
APÊNDICE A – Algoritmo do Dijkstra alterado…………………….………………... 63
APÊNDICE B – Módulos do arquivo created_init.h...……………….……………....65
APÊNDICE C – Módulos do arquivo router.h………………………..……………...69
APÊNDICE D – Módulo pirncipal do programa main.h…………………………...79
ANEXO A – Algoritmo do Dijkstra padrão………………...…………..…………..101

LISTA DE ILUSTRAÇÕES

Figura 1 - Forma de execução

Figura 2 - Tráfego de informações

Figura 3 - Dispositivos descarregados

Figura 4 – Rede particionada

Figura 5 - Gráfico de execução

Figura 6 - Tela de simulação do método de caminhos múltiplos

Figura 7 - Tela de execução do método de caminhos múltiplos

Figura 8 - Estrutura da composição de um sensor

Figura 9 - Tempo de execução para o algoritmo de Dijkstra

Figura 10 - Inicialização do algoritmo de Dijkstra

Figura 11 - Laço de repetição do algoritmo de Dijkstra

Figura 12 - Condição para aceitar um novo sensor e sua manipulação

Figura 13 - Adição do sensor a lista de visitados

Figura 14 - Ambiente de simulação

Figura 15 - Gráfico de energia

Figura 16 - Tela de execução

Figura 17 – Método de caminho único, tempo médio para o algoritmo de Dijkstra

Figura 18 – Método de caminhos múltiplos, tempo médio para o algoritmo de Dijkstra

Figura 19 – Número de ciclos 1

Figura 20 – Gráfico do número de ciclos 2

Figura 21 - Gráfico do número de dispositivos utilizados

Figura 22 - Gráfico de dados da execução de maior ciclo 1

Figura 23 - Gráfico de dados da execução 37

Figura 24 - Gráfico de dados da execução de maior pico 2

Figura 25 - Gráfico de dados da execução 8

Figura 26 - Gráfico da quantidade de caminhos encontrados

1. INTRODUÇÃO

Uma das maneiras de definir uma rede sem fio é através de comparações. Se houvesse uma troca da *Ethernet* cabeada por uma rede 802.11 (*wireless)* sem fio, o enlace de comunicação entre os hospedeiros, que antes era cabeada, agora seria sem fio. Um ponto de acesso pode substituir um comutador *Ethernet.* A camada de rede ou acima dela não seria necessário mudanças (KUROSE; ROSS, 2014).

Neste contexto a concentração estará voltada para a camada de enlace, pois nela seria possível encontrar diferenças significativas entre um enlace com fio e um enlace sem fio, como a redução da força do sinal, interferência de outras fontes e propagação multivias (KUROSE; ROSS, 2014).

A rede sem fio possui alguns elementos como:

- hospedeiro sem fio, que pode ser um notebook, smartphone, computador de mesa, entre outros, visto que a partir desses exemplos eles podem ser tanto dispositivos de mesa, quanto móvel;
- Enlace sem fio, é o enlace de comunicação sem fio, e as diferentes tecnologias aplicadas ao enlace influenciam na taxa de transmissão, podendo ter variações nesses valores;
- Estação base: definida como dispositivo responsável pelo envio e recebimento dos dados, coordenando a transmissão dos hospedeiros conectados a ela. Um exemplo prático é a torre de celular, e pontos de aceso em *Local Área Network* (LAN) sem fio 802.11 (KUROSE; ROSS, 2014).

Os dispositivos utilizados para construção da Rede de Sensores Sem Fio (RSSF) possuem limitações computacionais, em função disso, dificilmente conseguirá processar e realizar trabalhos de sensoriamento individual, porém em grande escala esses nós conseguem realizar tarefas mais extensas (GARAY, 2007).

Roteamento de caminho único e roteamento de caminhos múltiplos são dadas como as principais técnicas de roteamento. O primeiro é simples e escalável entretanto, não atua com eficiência em RSSF com limitação de recursos. É simples porque a rota

entre a origem e o destino pode ser estabelecida em um período de tempo específico. É escalável porque para uma rede com dez ou dez mil sensores, a complexidade e o método para descobrir o caminho permanece a mesma. (SHA et Cols, 2013, p. 2 e 3, tradução nossa).

Já o roteamento de caminhos múltiplos é uma alternativa que seleciona diferentes caminhos partindo de uma origem até o seu destino, e conforme o aumento dessa quantidade de Caminhos, aumenta o desempenho e a produtividade durante as transmissões. Ele é usado para realizar o balanceamento de carga ou para fornecer confiabilidade. As RSSF estão sujeitas a grandes taxas de falhas por conta de ruídos, obstáculos, mudanças ambientais, onde os sensores podem esgotar suas energias (GOPI, 2014, tradução nossa).

Considerando a situação em que se encontra a tecnologia, as RSSF podem ser aplicadas em locais que seja possível observar mudanças no ambiente em que se encontra, e reagir sobre elas (BEN-OTHMAN, 2010, tradução nossa). Por conta disso, essas redes são precisas em monitoramentos, tais como observar a temperatura, luminosidade, movimentações e presença de objetos (LIMA, 2019).

Diante dessa variedade de aplicações, voltar a atenção para a limitação dos recursos dos sensores pode ser essencial para projetar e implementar algoritmos eficientes, principalmente no que diz respeito a capacidade energética. Dessa forma, entregar dados com baixo consumo pode ser o ideal para essas redes.

O problema abordado se remete ao estudo sobre algoritmos de roteamento, e a sua implementação, incorporando a ele uma métrica que aponte o próximo melhor salto. Essa implementação esta baseada no algoritmo de Dijkstra, por ser um algoritmo abordado durante a graduação, e além disso, por ter sido publicado em 1959, e até hoje ainda ser utilizado.

O algoritmo de Dijkstra resolve o problema de caminhos mínimos de fonte única, citado por Cormen et Cols (2012), que a partir de um vértice origem dentro de um grafo, representado no Anexo B, é determinado um caminho mínimo a todos os demais vértices.

O estudo tem como objetivo geral comparar dois algoritmos de roteamento, o primeiro sendo de caminho único, e o segundo de caminhos múltiplos. Para alcançar esse fim, especificamente foi estudado o funcionamento das RSSF; foi feito uma análise e definida uma métrica dentro do algoritmo; e por fim investigou-se e discutiu-se os resultados obtidos.

Levando em consideração a abrangência do campo de pesquisa, este trabalho pode contribuir para estudos futuros e para aplicações que fazem uso das RSSF. A busca por informações é intensa, e métodos semelhantes já foram propostos, diante disso, este estudo aumenta o conjunto de informações disponíveis.

2. REFERENCIAL TEÓRICO

2.1 TRABALHOS SEMELHANTES

Diante das limitações dos sensores sem fio, que compõe a capacidade energética, poder de processamento e armazenamento, o campo de pesquisa para ter um melhor aproveitamento desses recursos é intenso.

Em Chen et Cols (2006), é proposto um protocolo de roteamento que se baseia no nível de energia atual dos sensores, ideia esta semelhante a adotada aqui neste estudo, porém no trabalho citado foi utilizado o modelo cliente/servidor, em que a estação base trabalha como um servidor, e os sensores trabalham como clientes, requisitando serviço e recebendo resposta.

O protocolo de roteamento de multicaminhos com reconhecimento de energia ou *Energy-aware Multipath Routing* (EMBR) proposto em Chen et Cols (2006), resolve problemas de algoritmos que encontram e utilizam uma única rota para comunicação, e acabam diminuindo a vida útil da rede. A ideia principal do EMBR é utilizar a estação base para encontrar múltiplas rotas, e escolher uma para utilizar na comunicação desejada, atualizando as energias, e selecionando novos caminhos.

O trabalho apresentado em Chen et Cols (2006), tem a sua atenção voltada para o tempo de vida útil da rede, ou seja, se preocupam com a energia dos sensores, ideia que compõe o presente estudo.

Em Arora et Cols (2013) são propostos dois algoritmos de roteamento, sendo o primeiro de caminho único e o segundo de caminhos múltiplos, idéia esta semelhante ao estudo. No primeiro algoritmo é utilizada a transmissão de dados através de um único caminho, até que a força do caminho caia ou se esgote, então outra rota disponível será utilizada. Já a segunda versão, os dados são transferidos através de múltiplas rotas, e para cada rota é atribuído a probabilidade de sucesso na entrega dos dados.

A busca pelos caminhos realizada pelo algoritmo proposto em Arora et Cols (2013) é baseada em três fatores: A energia residual dos sensores, tamanho do *buffer* disponível, e a qualidade do sinal. Assume-se que os sensores são idênticos. No estudo atual, utiliza-se uma métrica similar a força mencionada, pois é uma relação entre a energia residual dos dispositivos, e a distância entre os seus vizinhos.

Em Gopi (2014), é realizada uma análise sobre o roteamento de caminhos múltiplos, onde um dos conceitos abordados é utilizado aqui no presente estudo. O processo de descoberta de caminhos pode ser definida como:

- caminhos múltiplos com nós disjuntos, em que o conjunto de rotas não possuem sensores em comum;
- caminhos múltiplos com link disjuntos, que não existe compartilhamento de links entre as rotas, mas pode haver dispositivo em comum;
- e por fim caminhos múltiplos parcialmente disjuntos, em que o conjunto de rotas pode compartilhar vários links ou dispositivos entre rotas diferentes.

O modelo de descoberta de caminhos do trabalho aqui apresentado, é descrito pelo modelo de caminhos múltplos com nós disjuntos.

Em Sha et Cols (2013) é apresentada uma pesquisa sobre as técnicas de roteamento de caminhos múltiplos para redes de sensores sem fio, abordando conceitos pertinentes para elaboração do método. Roteamento de caminho único e roteamento de caminhos múltiplos, são as principais técnicas de roteamento, o primeiro é simples e escalável, entretanto não atua com eficiência em RSSF com limitação de recursos.

As técnicas de roteamento de caminhos múltiplos são classificadas como: 1) Baseado em infraestrutura, 2) Não baseado em infraestrutura e 3) Baseado em codificação. A categoria 1 constrói e mantém uma infraestrutura específica, já na categoria 2, não é utilizado uma infraestrutura específica, e o seu próximo salto é decidido com base em seu conhecimento local. A categoria 3 utiliza esquemas de

codificação para fragmentar o pacote de dados na origem, e em seguida envia-los (SHA et Cols, 2013, p. 2 e 3, tradução nossa).

Em relação ao trabalho atual, ele pode ser classificado como a categoria baseado em infraestrutura, já que em intervalos de tempos o algoritmo de Dijkstra é executado, e o caminho de uma origem para um destino já estará definido se existir, mesmo que a origem e o destino ainda não estejam definidos.

Um protocolo de roteamento de caminhos múltiplos baseado em qualidade de serviço e eficiência energética é proposto em Ben-Othman (2010), esse algoritmo alcança o balanceamento de carga por meio de múltiplos caminhos com dispositivos internamente disjuntos. É um protocolo que possui similaridade com o estudo aqui apresentado, que utiliza a ideia de caminhos múltiplos com nós internamente disjuntos.

Semelhante ao que foi apresentado em Arora et Cols (2013), neste trabalho Ben-Othman (2010) assume que os sensores são idênticos, e distribuídos aleatoriamente, sendo capazes de calcular o valor da sua energia residual, o tamanho do *buffer,* e registrar a relação de sinal ruído. Cada caminho encontrado, está associado a uma probabilidade, para assim selecionar as rotas que possam garantir de fato a entrega dos pacotes de dados.

2.2 FERRAMENTAS

2.2.1 LINGUAGEM C

A linguagem C é uma linguagem estruturada, considerada de nível médio, mas não significa que ela é menos poderosa ou menos desenvolvida que as linguagens de alto nível, mas também não é tão complicada como assembly. C é considerada de nível médio por combinar elementos de linguagem de alto nível com funcionalidades do assembly (SHILDT, 1996).

Essa linguagem permite a criação de listas encadeadas, utilizando ponteiros, sendo possível alocar e liberar memória quando necessário. Como por exemplo a fila de prioridade mínima, criada para auxiliar o algoritmo de Dijkstra, e as demais listas. Existe fóruns, blogs e tutoriais grátis na internet que podem auxiliar o programador em qualquer nível de dificuldade.

Além disso, como é uma linguagem de médio nível, não possui muita função ou estrutura pronta, como em Python e Java, e se por um lado o programador "perde" seu tempo desenvolvendo funções que já estão implementadas em outras ferramentas, isso faz com que ele também desenvolva apenas o suficiente para sua aplicação, sem ter contato com estruturas da própria linguagem, que possuem bastante funcionalidade.

2.2.2 OPENGL

OpenGL é uma biblioteca gráfica que pode ser manuseada em diversos sistemas operacionais. Utilizada para modelagem e exibições em terceira dimensão, possuindo recursos que permitem a criação de objetos com qualidade e de modo rápido, além de incluir recentes recursos de animação, tratamento de imagens e textura. OpenGL É uma biblioteca gráfica de modelagem e exibição tridimensional, bastante rápida e portável para vários sistemas operacionais. Seus recursos permitem ao usuário criar objetos gráficos com qualidade próxima à de um *raytracer*, de modo mais rápido que este último, além de incluir recursos avançados de animação, tratamento de imagens e texturas (CARVALHO, 2006).

Essa biblioteca pode ser manuseada utilizando a linguagem C, com ela é possível exibir círculos que podem representar os sensores, utilizando cores para distinguir um do outro. Além disso possui uma documentação interessante e também há a presença de fóruns e comunidades ativas que auxiliam no desenvolvimento.

2.2.3 GNUPLOT

O gnuplot é uma ferramenta para criação de gráficos a partir de linhas de comando em um terminal Linux, OS/2, Windows, OSX, VMS entre outras plataformas, sendo possível visualizar graficamente dados coletados (GNUPLOT, 2019)

A partir da API (*application programming interface*) do gnuplot para linguagem C, é possível inseri-lo dentro do projeto, para que durante a execução seja possível analisar alguma(s) variável de interesse, como a energia, o tempo, o número de rotas encontradas, entre outros. A documentação disponibilizada é de fácil aprendizado, com demonstrações e inglês técnico.

3. METODOLOGIA

Aplicar o conhecimento científico neste estudo é de suma importância, pois pretende-se buscar explicações para anomalias e tirar conclusões dos resultados obtidos perante a comparação entre os métodos de roteamento. Diante desse fato, será utilizado no estudo um *notebook* de uso pessoal, com um compilador GCC e o sistema operacional Linux, para fins de testes.

A metodologia utilizada aqui nesse trabalho primeiramente correspondeu ao estudo e análise da literatura, buscando conhecer os componentes de uma RSSF, bem como o funcionamento do algoritmo de Dijkstra, para isso, os livros “Algoritmos: Teoria e Prática. (CORMEN et Cols, 2012) e “Redes de computadores e a internet - uma abordagem top-down. (KUROSE e col, 2014)”, deram suporte para esses temas. Para tópicos mais específicos sobre os métodos de roteamento, e suas características, os artigos e periódicos forneceram o conhecimento necessário para realização desse trabalho.

Durante a fase de desenvolvimento, foram implementados utilizando a linguagem C, os métodos de roteamento de caminho único e caminhos múltiplos, com o auxílio do orientador, mediante as reuniões, e discussões, corrigindo o algoritmo quando necessário.

Após os ajustes efetuados no algoritmo, realizou-se a coleta de dados, de um conjunto de 100 execuções, para efetuar análise de tempo de execução do algoritmo de Dijkstra, bem como o desempenho da bateria dos sensores, o número de ciclos até que a energia da rede se esgote, o número de dispositivos utilizados, e a quantidade de rotas encontradas pelo método de caminhos múltiplos.

O método comparativo é um dos métodos de procedimentos existentes e foi utilizado no presente estudo, aplicando-o sob ambos protocolos, para analisar o desempenho de cada um.

Conforme finalizou-se a implementação dos métodos e realizada a coleta de dados, foi elaborada a escrita do trabalho, apresentando os componentes da RSSF, os desafios desse campo, o funcionamento do algoritmo de Dijkstra, os métodos de roteamento e técnicas, além de outros tópicos que deram suporte ao desenvolvimento do projeto.

4. DESENVOLVIMENTO

Nesta sessão é apresentado o processo de execução dos métodos, bem como suas fases, desde o início ao fim da execução. Trechos de códigos demonstrando a captura de dados, e o processo de roteamento também será assuntos desta sessão. As telas aqui apresentadas não fazem relações uma com as outras, ou seja, são de execuções diferentes, exibindo estados diferentes, apenas para fins educativos.

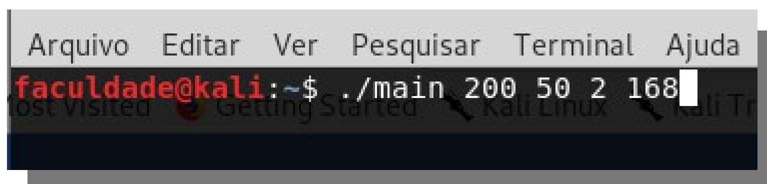

Figura 1 - Forma de execução

Fonte: Elaborado pelo autor, 2019

A Figura 1 acima, representa a forma de execução da aplicação, em que o "./main" representa o nome do arquivo executável. Já o 1º parâmetro "200", representa o número de dispositivos que será gerado durante a simulação. O 2º parâmetro "50", indica a unidade de medida máxima de distância para que dois dispositivos possam estar conectados. Já o 3º e 4º parâmetro "2" e "168", representam respectivamente o dispositivo de origem, e o dispositivo de destino, dessa forma o usuário terá conhecimento da localidade desses dispositivos somente no momento em que a simulação iniciar.

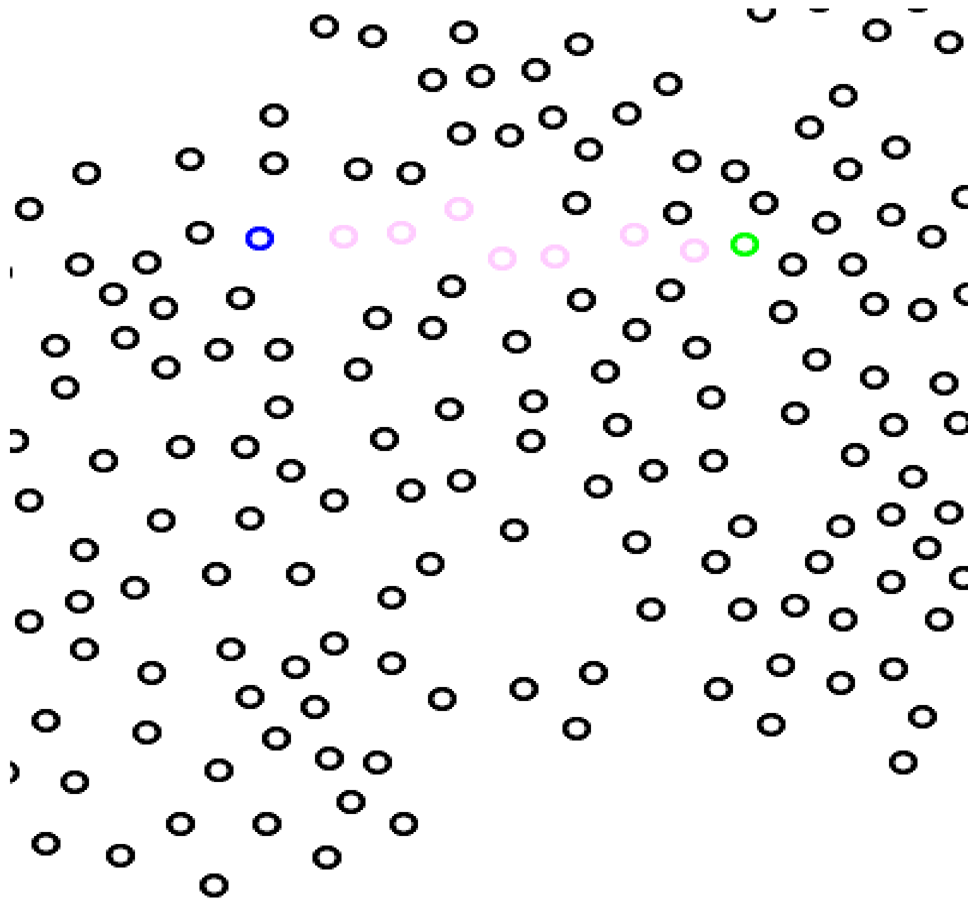

Figura 2 - Tráfego de informações

Fonte: Elaborado pelo autor, 2019

Na Figura 2 é apresentado o caminho para tráfego de informações. Representado pela cor verde, tem-se a origem dos dados, e representado pela cor azul o receptor da informação. Já os dispositivos de cor rosada, representam os nós intermediários da comunicação entre a origem e o destino, e os de cor preta representam os demais sensores da rede que não estão sendo utilizados.

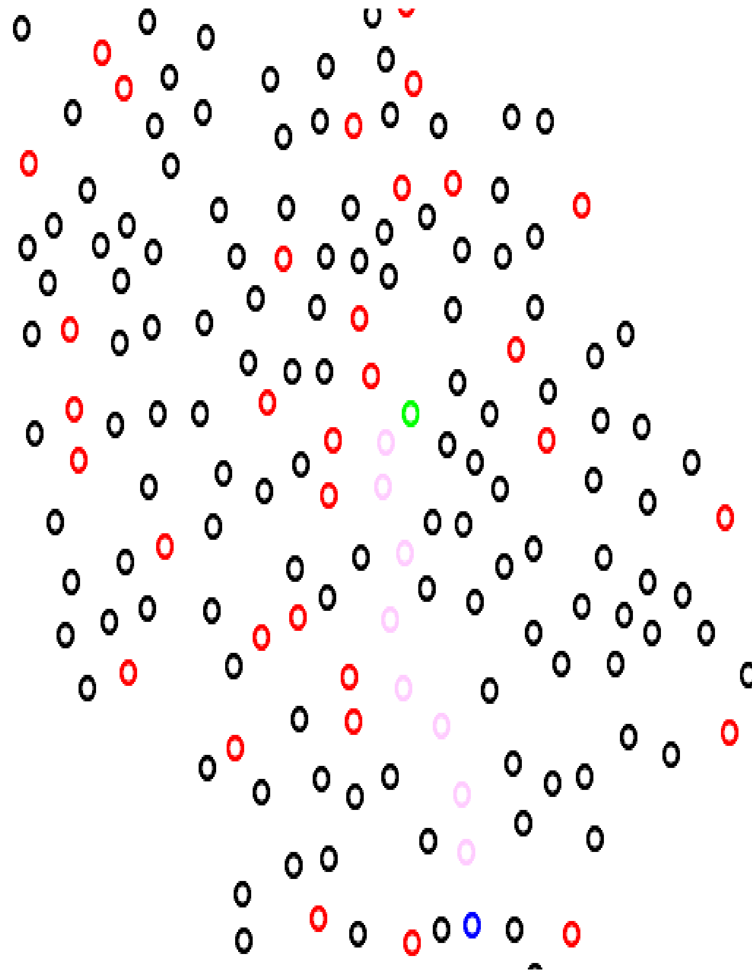

Figura 3 - Dispositivos descarregados

Fonte: Elaborado pelo autor, 2019

Já na Figura 3, apresenta-se mais uma vez o tráfego de informações, porém dessa vez temos dispositivos descarregados, representados pela cor vermelha.

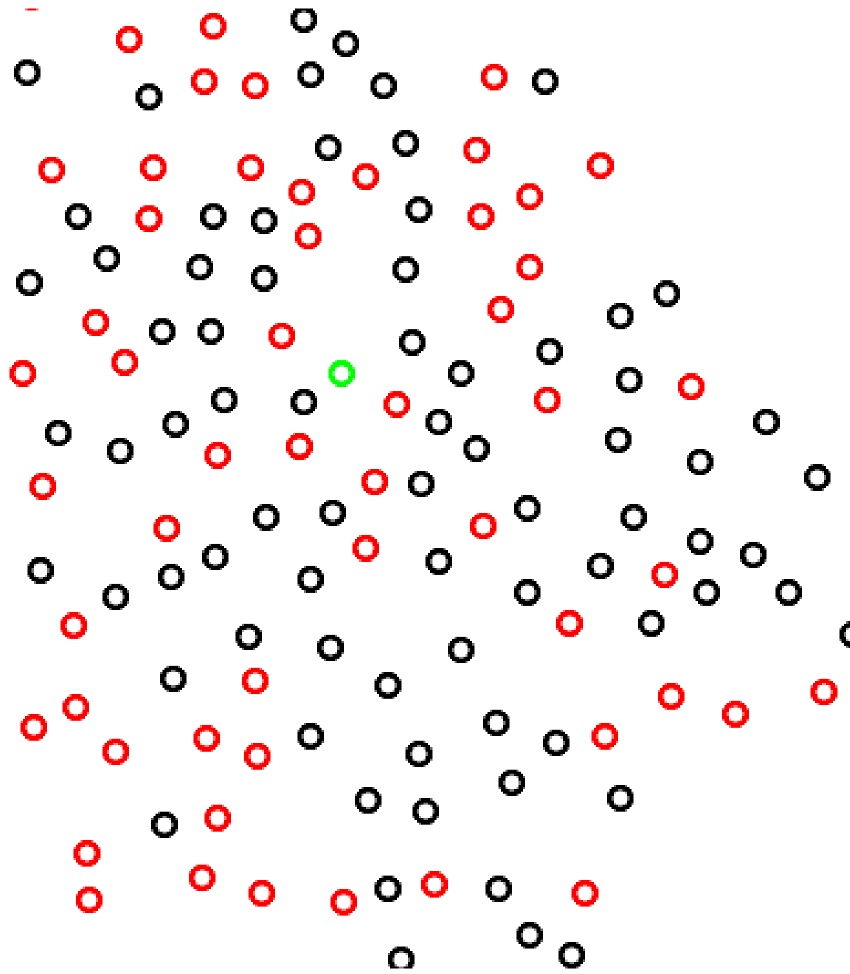

Figura 4 – Rede particionada

Fonte: Elaborado pelo autor, 2019

Na figura 4 tem-se a rede particionada, sem a representação dos dispositivos intermediários da origem até o destino, pois a rede não é mais conexa, e a origem e o destino pertencem a partições diferentes.

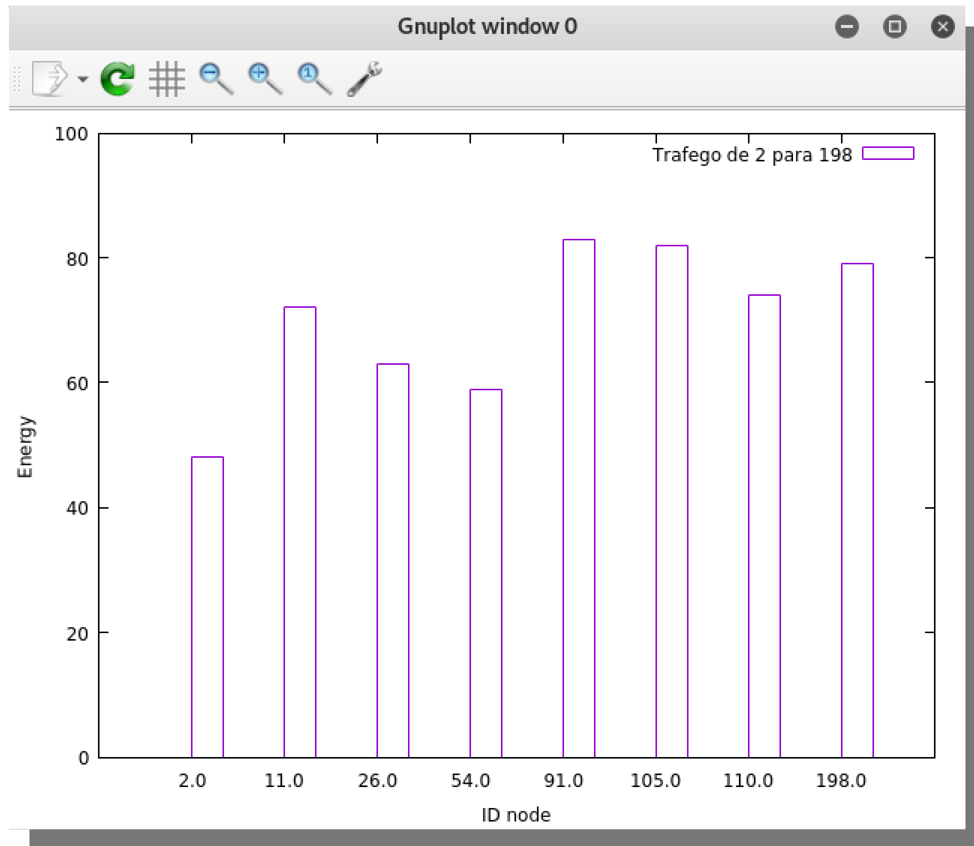

Figura 5 - Gráfico de execução

Fonte: Elaborado pelo autor, 2019

A Figura 5, apresenta um gráfico de energia (representado pelo eixo Y “Energy”) dos dispositivos que pertencem a rota encontrada, demonstrado no gráfico pelo eixo X “ID node”. Durante a execução dos métodos de roteamento, além de simular o roteamento, esse modelo de gráfico também é exibido. Vale ressaltar que o método de multicaminhos, também utiliza esse mesmo modelo de gráfico e pode apresentar mais de um, já que o mesmo não busca somente um caminho, mas vários, dessa forma será gerado um gráfico para cada rota encontrada.

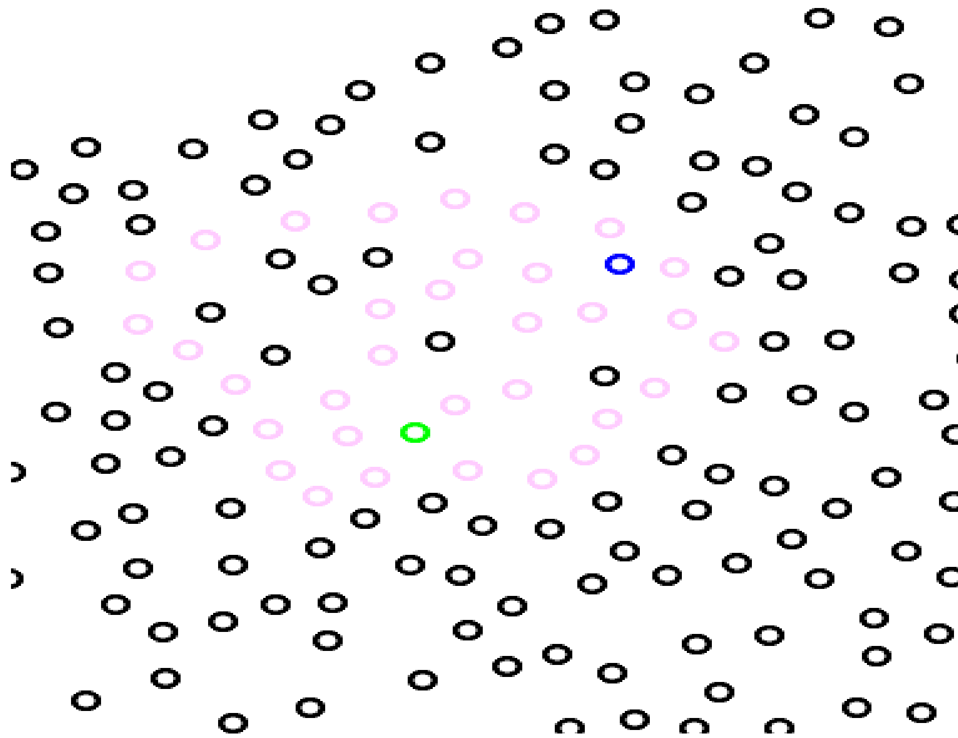

Figura 6 - Tela de simulação do método de caminhos múltiplos

Fonte: Elaborado pelo autor, 2019

Na Figura 6 é apresentada a tela de simulação do método de caminhos múltiplos, que segue a mesma representação de cores padrões da tela de caminho único abordado na exibição da Figura 2. Esta tela se diferencia por não possuir apenas um caminho, mas sim vários, apresentando aqui especificamente quatro rotas.

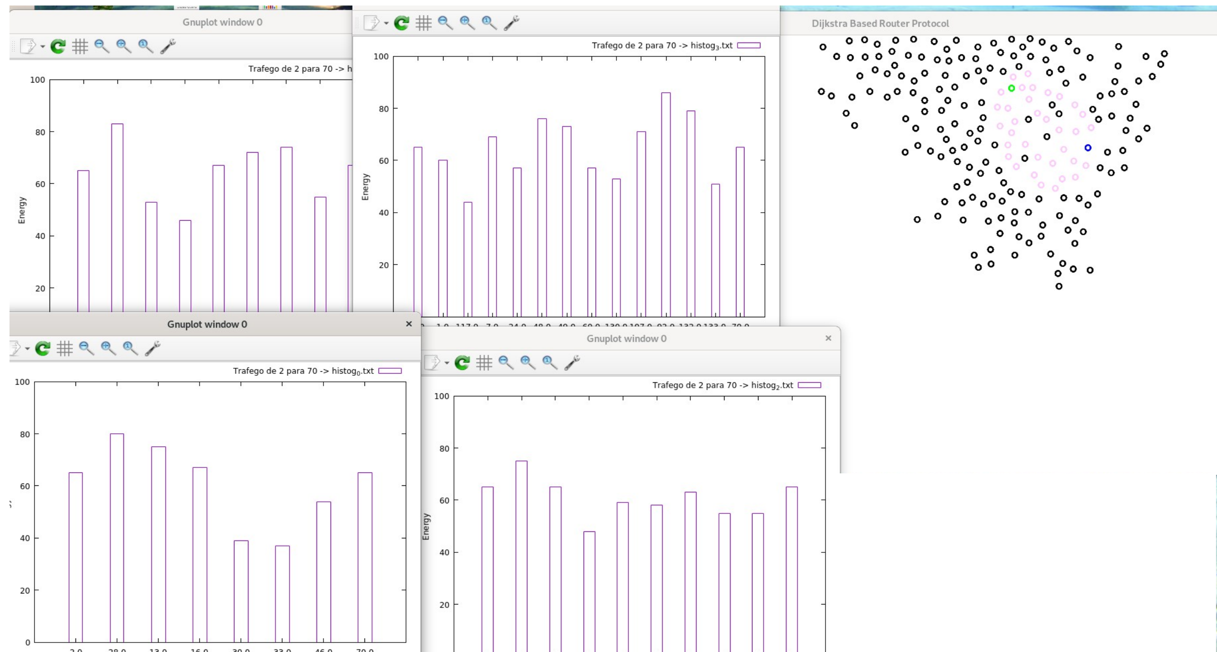

Figura 7 - Tela de execução do método de caminhos múltiplos

Fonte: Elaborado pelo autor, 2019

A Figura 7 apresenta ao leitor a tela final de execução do algoritmo de caminhos múltiplos, exibindo em conjunto a simulação apresentada na Figura 6, contendo quatro rotas, e os gráficos que indicam a quantidade de energia que os sensores possuem, que segue o mesmo modelo exibido na Figura 5. Nesta tela são mostrados quatro gráficos pois está ligado ao número de rotas, ou seja, foram encontradas quatro rotas.

A tela final de execução do método de caminho único é demonstrada na Figura 16 da sessão 5.

```
int Id;/*id = <posicao_vetor>*/
int Pwi; /*Energia inicial do sensor. Gerada entre 70 e 99 */
int Pwa; /* Energia atual, Gerada entre Pwi e (Pwi - 30)*/
```

(a) Atributos de identificação e de energia

```
/*x e y representa a posicao do sensor no p
int x;
int y;
```

(b) Coordenadas de localização

```
float **mtAdjMet; /*matriz [2][N], onde na primeira linha temos os sensores adjacentes com
sua respectiva distancia ao sensor referencia. E na segunda linha temos as metricas para esses sensores.*/

float **mtDijks; /*representa a matriz gerada quando o algoritmo de dijkstra eh executado */
```

(c) Matrizes auxiliares

```
int color; /*Representa as cores dos dispositivos na parte grafica
            0 - preta       - dispositivo qualquer
            1 - verde       - representa o inicial
            2 - azul        - representa o final
            3 - rosado      - participa da comunicacao fim a fim
            4 - vermelho     - dispositivo sem carga
            */

int sendData; /* Essa variavel auxilia no descobrimento de multicaminhos
                1 - se enviou dados
                 0 - caso nao envou*/

int countRoutes; /*representa a quantidade de rotas encontradas */
```

(d) Atributos de estado e quantificador de rotas

Figura 8 - Estrutura da composição de um sensor

Fonte: Elaborado pelo autor, 2019

A Figura 8 apresenta ao leitor, os atributos que foram necessários para formar a estrutura de um sensor, sendo a mesma estrutura para ambos os métodos, exibindo atributos como o identificador, definido como **Id** (Figura 8.a), que representa também a posição do sensor dentro do vetor (estrutura onde é armazenado todos os demais sensores).

Na mesma figura também são apresentados os atributos que correspondem a energia dos sensores, em que **Pwi** corresponde a energia inicial, ou seja quando o dispositivo está teoricamente totalmente carregado (nem sempre a bateria do

dispositivo indicando 100% de fato ela está com essa carga, por isso é gerado randomicamente), e o **Pwa**, que representa a energia atual com o em um determinado momento, já que a energia é reduzida com o passar do tempo.

A Figura 8.b mostra os atributos que correspondem as coordenadas de localização do sensor em um plano. Isso é necessário pois a bilbioteca gráfica do OpenGL trabalha com o plano de 2 dimensões, sendo assim, as coordenadas x e y são necessárias para realizar a plotagem.

Já na Figura 8.c as matrizes auxiliares para cada sensor são apresentadas. O atributo **mtAdjMet** representa uma matriz com duas colunas, onde é armazenada a distância até o seu vizinho, bem como o resultado de cálculo da métrica. A **mtDijks** representa a matriz gerada durante a execução do algoritmo de Dijkstra, que diferente do algoritmo padrão, não possui as distâncias até o nó, mas sim as métricas. É também anexada a matriz, mais duas colunas, que representam o predecessor do sensor, apenas para manter uma informação que é gerada durante a execução do Dijsktra, e quem é o seu sucessor, ou seja, qual deve ser o próximo salto para alcançar um destino X.

Por fim, na Figura 8.d, apresenta-se **color** que define a cor do sensor a ser exibida durante a simulação, visto que cada cor representa um tipo de dispositivo. **sendData** que permite verificar se o sensor a ser analisado já participa de uma rota ou não e o **countRoutes** que calcula quantas rotas foram encontradas pelo algoritmo.

```
gettimeofday(&time_inicial, NULL);
tIni = (double) time_inicial.tv_usec / 1000000 + (double) time_inicial.tv_sec;
for (i = 0; i < N; i++) {
    if(Nos[i].Pwa > 0)
        dijkstra(&Nos[i], Nos, N);
}

gettimeofday(&time_final, NULL);
tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;
```

Figura 9 - Tempo de execução para o algoritmo de Dijkstra

Fonte: Elaborado pelo autor, 2019

A Figura 9, apresenta a coleta de tempo de execução do algoritmo de Dijkstra, utilizando a função **gettimeofday,** que é definida na biblioteca *sys/time.h,* passando como parâmetro uma estrutura do tipo *struct timeval,* que permite no momento da sua chamada, recuperar o tempo em segundos, após efetuar os cálculos apresentados na própria Figura 9. Dessa forma, conseguiu-se a partir da coleta de tempo antes do início da execução do algoritmo, e após o fim da sua execução, calcular a duração de tempo.

```
inicia_d_p(d, p, INI->Id, N);/*inicializacao semelhante ao dijkstra padrao*/
FILA = NULL;
visitados = NULL;
adc_fila(&FILA, INI->Id, d[INI->Id]);
```

Figura 10 - Inicialização do algoritmo de Dijkstra

Fonte: Elaborado pelo autor, 2019

Conforme o algoritmo de Dijkstra padrão, o algoritmo modificado também possui a inicialização das estruturas **d** e **p**, do sensor inicial, e dos demais dispositivos da rede, sendo representada na Figura 10, em que o **d** representa a métrica calculada para alcançar o dispositivo, e o **p** representa o melhor predecessor conforme o cálculo da métrica. O atributo **FILA** representa a fila de prioridade mínima, ou seja, aquele dispositivo que estiver com a menor métrica, será o primeiro a sair da fila durante a execução do algoritmo. A variável **visitados** representa a lista de dispositivos já visitados pelo Dijkstra. Ambas variáveis **FILA** e **visitados** são estruturas do tipo lista encadeada.

```
while(FILA != NULL){
    u = remove_min(&FILA); /*remove da fila o sensor de menor metrica */
    pos_u = u->id; /*salvo o identificador dele, que representa a sua posicao no vetor */
    for(i = 0; i < N; i++){
        if(conjNos[u->id].mtAdjMet[1][i] > 0.0 /*metrica maior que 0 sao vizinhos */ && conjNos[u->id].mtAdjMet[1][i] < 100.0
```

(a) - Laço de repetição incompleto

```
&& !pertenceVisitadosFila(visitados, &conjNos[i]) && conjNos[i].sendData == 0)
```

(b) - Complemento do laço de repetição

Figura 11 - Laço de repetição do algoritmo de Dijkstra

Fonte: Elaborado pelo autor, 2019

A Figura 11 apresenta o início do laço de repetição para todos os dispositivos. Inicialmente na Figura 11.a mostra que enquanto houver algum dispositivo na **FILA** o processo continua executando, removendo o primeiro sensor da fila e inserindo-o em **u,** recuperando a sua posição no vetor que contém todos os sensores, guardando esse valor em **pos_u.**

Após o armazenamento da posição, inicia-se a execução de outro laço para cada dispositivo da rede, que seja adjacente ao atual, para isso ele deve possuir uma métrica entre 0 e 100, pois a métrica está regulada para valores entre 0 e 1, e menor que 100, valor atribuído quando não há possibilidade de cálculo para esse dispositivo, visto que não são adjacentes.

Esse alto limite máximo permite incluir métricas com valores fora da regularidade desejada (0 e 1), controlando apenas os cálculos que de fato não foram possíveis. A consulta é realizada utilizando o **conjNos,** que representa o vetor onde estão guardados todos os sensores.

Já o complemento da condição do laço mais interno apresentado na Figura 11.b, complementa a condição **if,** verificando se o dispositivo a ser analisado não pertence a fila de visitados, chamando a função **!pertenceVisitadosFila** que retorna 0 caso o dispositivo não pertença, caso contrário retornará 1. Para finalizar, verifica se o mesmo dispositivo não participa de nenhum outro caminho, se ele já participa **sendData** é igual a 1, caso contrário 0.

```
if(d[i] > d[pos_u] + metrica){
    if(p[i] != -1){//aqui significa que ele possuia um pai, e o seu pai deverá ser corrigido
        //pois uma distância menor foi encontrada
        aux = p[i];
        INI->mtDijks[aux][i] = 0.0;
    }
    if(u->id == INI->Id){//se for o primeiro que estamos analisando, o destino são os próprios filhos.
        //A posição N+1 é o destino do pacote
        INI->mtDijks[i][N+1] = i;
    }else{//o destino do pacote será o mesmo destino do pai dele, ou seja, o nó atual que está analisando
        INI->mtDijks[i][N+1] = INI->mtDijks[pos_u][N+1];
    }
```

(a) - Parte 1

```
INI->mtDijks[i][N] = u->id;//identifica o predecessor dele, (FAMOSO PI DE U)

//atualiza as variáveis internas da função
d[i] = d[pos_u] + metrica;
p[i] = pos_u;

INI->mtDijks[pos_u][i] = d[i];//atualiza a matriz de u, adicionando a distancia acumulada para o destino i

se_existe_remove(&FILA, i);

adc_fila(&FILA, i, d[i]);//adiciona i na fila, (i representa não só a posição mas também é o identificador do nó)
```

(b) - Parte 2

Figura 12 - Condição para aceitar um novo sensor e sua manipulação

Fonte: Elaborado pelo autor, 2019

A Figura 12.a apresenta a condição para aceitar um vizinho ainda não visitado **i**, verificando se a métrica atual do dispositivo adjacente **d[i]** é maior do que a métrica do sensor atual a ser analisado **d[pos_u]** somado com a métrica para alcançar **i,** o atributo **metrica**. Em caso afirmativo, é verificado se o sensor **i** já possui um predecessor, pois se possuir será atualizado a tabela de metricas do sensor **pos_u**, retirando a conexão entre **p[i]** e o próprio **i.** Caso **u** for o dispositivo inicial, definido por **INI,** a interface de saída da tabela do dispositivo inicial, será o próprio dispositivo **i,** caso contrário, será a mesma interface do dispositivo que está sendo visitado (**pos_u**).

Já a figura 12.b, apresenta a segunda parte da manipulação de dados, inserindo o predecessor do dispositivo **i** na tabela de Dijkstra do dispositivo inicial. E atualiza as estruturas **d[i]** e **p[i],** incrementando a métrica e alterando o predecessor de **i.** Ocorre também, a atualização dentro da tabela de Dijkstra, do valor da métrica de **u** para alcançar **i** a partir de **INI,** então é verificado na fila se existe o dispositivo de identificador **i** para removê-lo, e então o adiciona a fila de prioridade mínima.

A métrica aqui mencionada corresponde ao seguinte cálculo:

$$Metrica(x,y)=\frac{K\cdot d(x,y)^2}{\sqrt[3]{\frac{Pwa_y}{Pwi_y}}}$$

Em que a *Metrica(x,y)* corresponde a métrica calculada para o dispositivo *x*, alcançar *y*. *K* representa uma constante para ajuste de valores; *d(x,y)* se refere a distância entre *x* e *y;* e Pway corresponde a energia atual do sensor *y,* já o Pwiy é a energia inicial do dispositivo quando inserido na rede.

```
AdcListaVisitados(&visitados, u);
```

Figura 13 - Adição do sensor a lista de visitados

Fonte: Elaborado pelo autor, 2019

Por fim, ao analisar todos os vizinhos de **u**, ele próprio é adicionado na fila de visitados, para que não possa ser mais analisado, como é exibitdo na Figura 13.

5. ANÁLISE E INTERPRETAÇÃO DOS RESULTADOS

Após a implementação dos dois métodos, foi realizado a coleta de dados, executando o roteamento com 700 dispositivos, partindo do sensor com identificador igual a 2 para o 489, executando cada um dos algoritmos uma quantia de cem vezes.

Nesta sessão, são apresentadas as telas do sistema, a representação gráfica dos dados coletados, e a sua descrição.

5.1 TELAS

As telas aqui apresentadas, representam o ambiente de simulação da aplicação, em específico o método de caminho único. Para essa demonstração, foi gerada uma rede com 200 sensores, com o objetivo de melhorar a visualização do leitor.

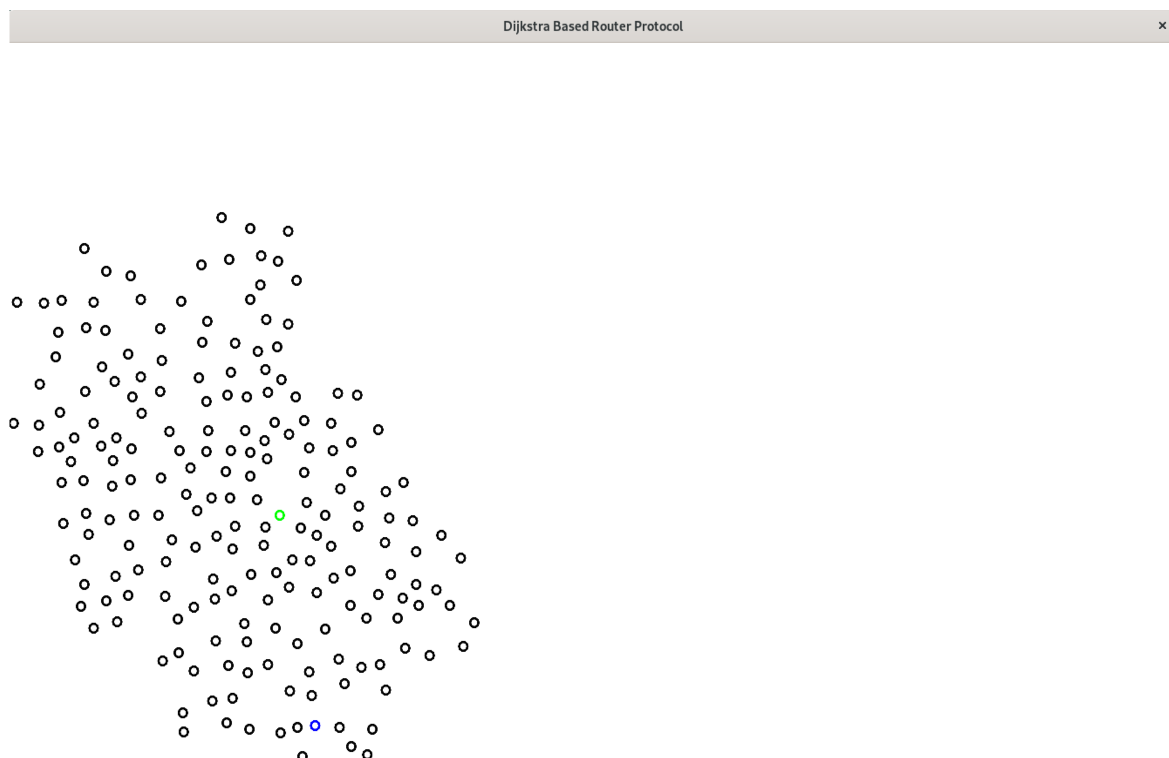

Figura 14 - Ambiente de simulação

Fonte: Elaborado pelo autor, 2019

A Figura 14 representa o ambiente de simulação do método de roteamento de caminho único, apresentando nesta tela através de circunferências os dispositivos. O caminho encontrado pelo método também é exibido neste momento, em que o dispositivo de cor verde representa a origem dos dados, os rosados representam os dispositivos em que os dados transitaram, e o azul sendo o destino dos dados. Os dispositivos de cor preta fazem parte da rede e estão disponíveis, porém não foram escolhidos.

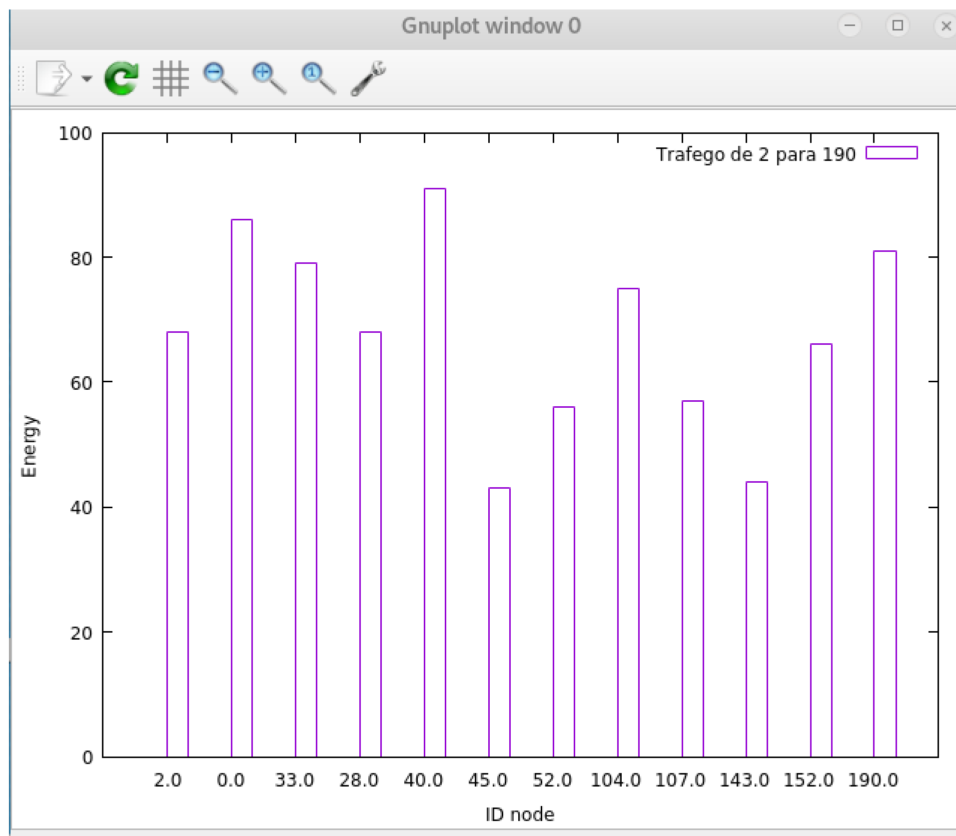

Figura 15 - Gráfico de energia

Fonte: Elaborado pelo autor, 2019

O gráfico apresentado na Figura 15, acompanha a Figura 14, caracterizando a energia residual dos dispositivos que fazem parte da rota encontrada, desde a origem até o destino. Neste gráfico (Figura 15) o eixo X comporta os identificadores dos sensores em sequência, partindo do remetente até o destinatário. O eixo Y apresenta a energia residual desses dispositivos.

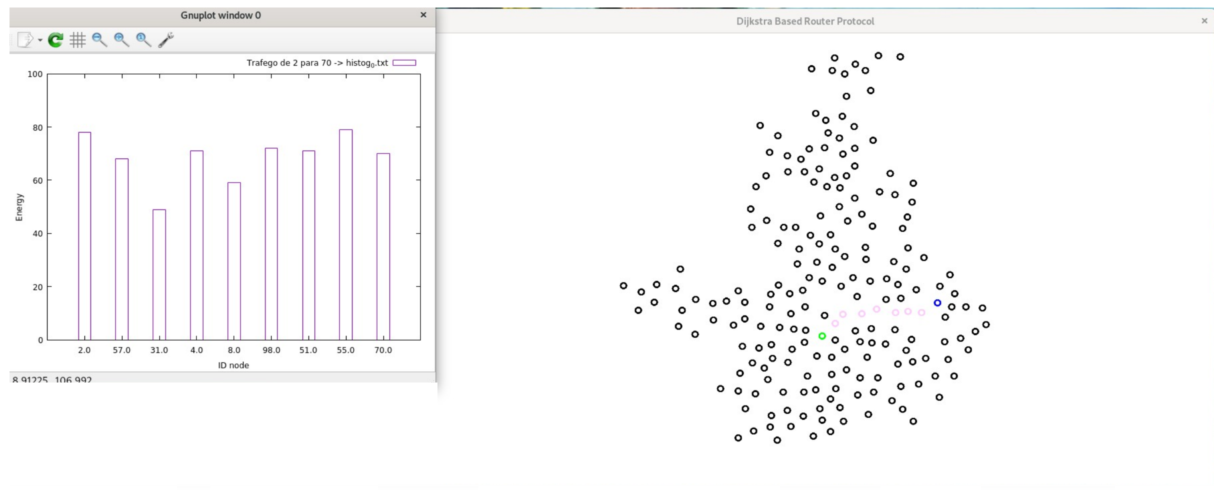

Figura 16 - Tela de execução

Fonte: Elaborado pelo autor, 2019

Por fim, a Figura 16 apresenta em conjunto, a tela final após a execução do programa, sendo a mesma composta por uma simulação de roteamento, e um gráfico de energia, abordados na apresentação da Figura 14 e 15.

5.2 REPRESENTAÇÃO GRÁFICA DOS RESULTADOS

Nesta sessão são apresentados os dados coletados em forma gráfica, seguido da sua descrição e comparações.

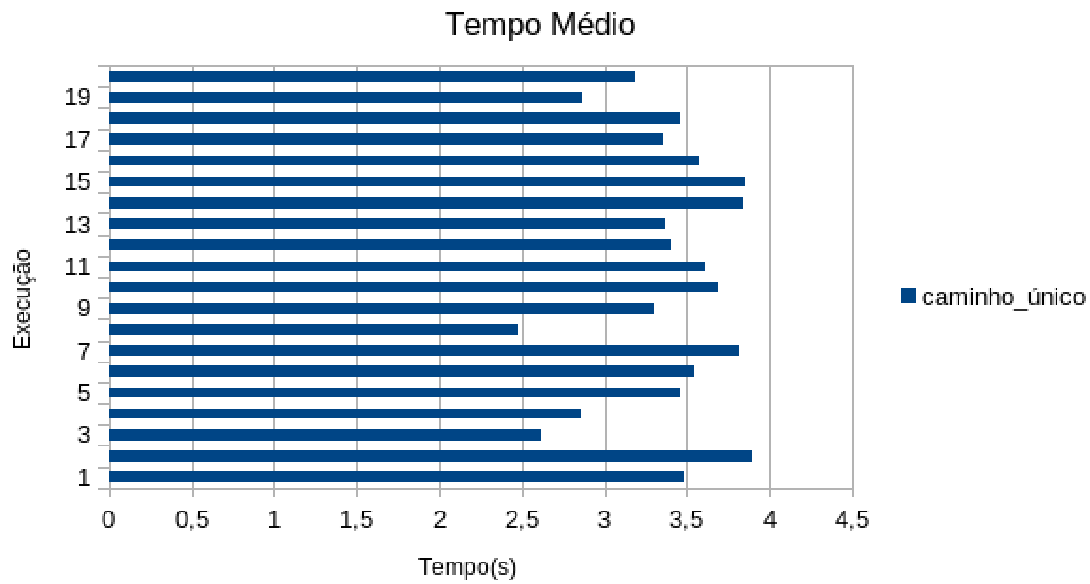

Figura 17 – Método de caminho único, tempo médio para o algoritmo de Dijkstra

Fonte: Elaborado pelo autor, 2019

A Figura 17, representa uma amostra do tempo de execução do algoritmo de Dijkstra modificado, para o método de caminho único, apresenta algumas variações pois para cada execução uma nova topologia é gerada, o que pode diminuir ou aumentar o tempo de busca. Para essas mesmas topologias, foram realizadas coletas de tempo para o método de caminhos múltiplos, como segue a Figura 18 abaixo.

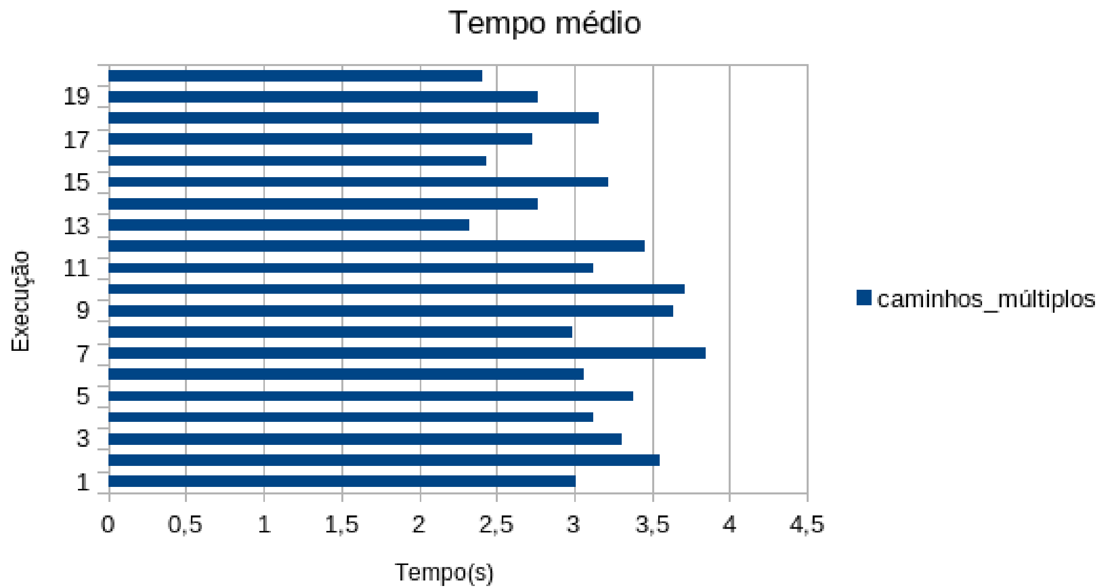

Figura 18 – Método de caminhos múltiplos, tempo médio para o algoritmo de Dijkstra

Fonte: Elaborado pelo autor, 2019

Para o método de caminhos múltiplos, como consta a Figura 18, o intervalo entre a execução 13 e 19, todos os tempos estão abaixo de 3,5 segundos, e na Figura 17 alguns valores estão próximos e outros ultrapassam. Isso se dá pelo fato do método de caminho único analisar uma grande quantidade de sensores apenas uma vez, já para o algoritmo de caminhos múltiplos apenas a primeira vez antes de encontrar a primeira rota isso acontece, já para as próximas execuções do algoritmo de Dijkstra, alguns sensores não precisam de análise, pois já fazem parte de uma rota.

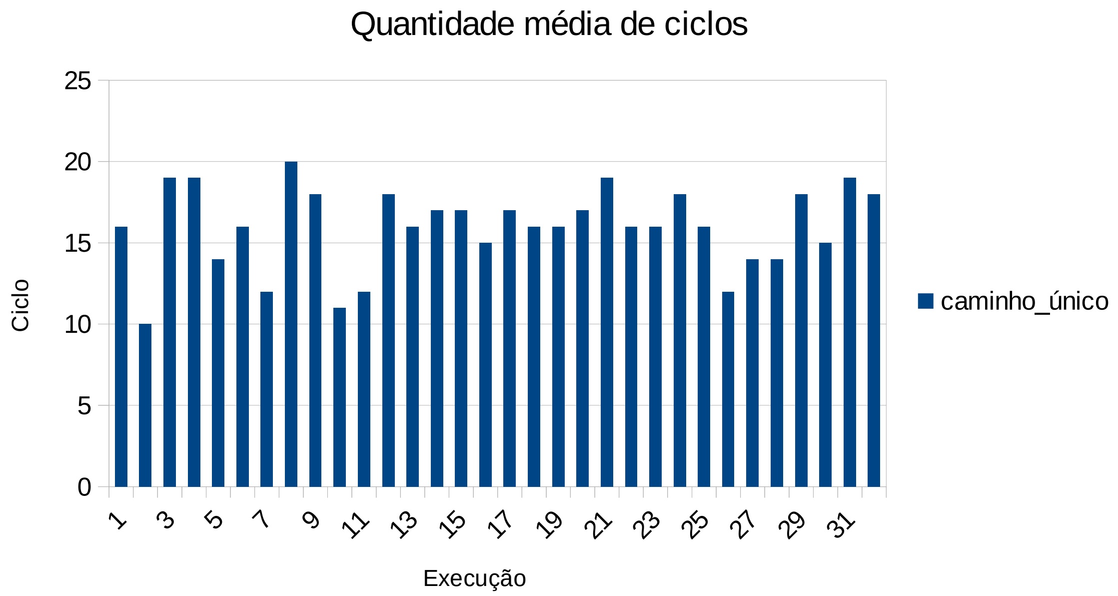

Figura 19 – Número de ciclos 1

Fonte: Elaborado pelo autor, 2019

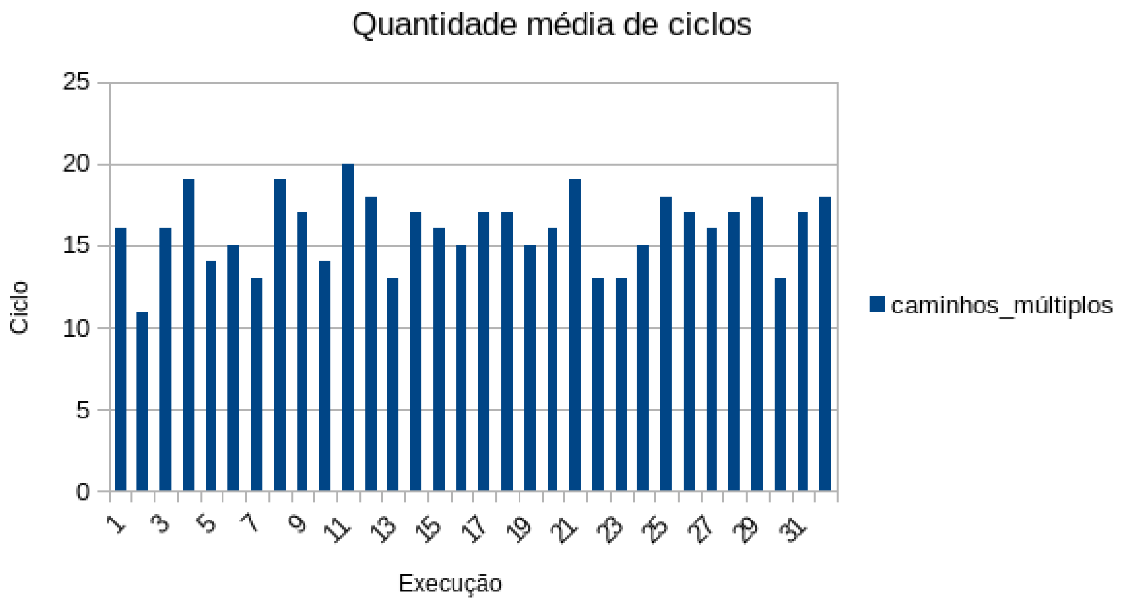

Figura 20 – Gráfico do número de ciclos 2

Fonte: Elaborado pelo autor, 2019

É apresentado nas Figuras 19 e 20, a quantidade média de ciclos de ambos algoritmos, mostrando que esses métodos, submetidos as mesmas condições ambiente em uma mesma topologia, acabam por se assemelhar nessa questão, que representa a quantidade de vezes que o algoritmo conseguiu realizar o roteamento e encontrar pelo menos uma rota, pois conforme o passar do tempo a energia dos sensores sofrem uma queda.

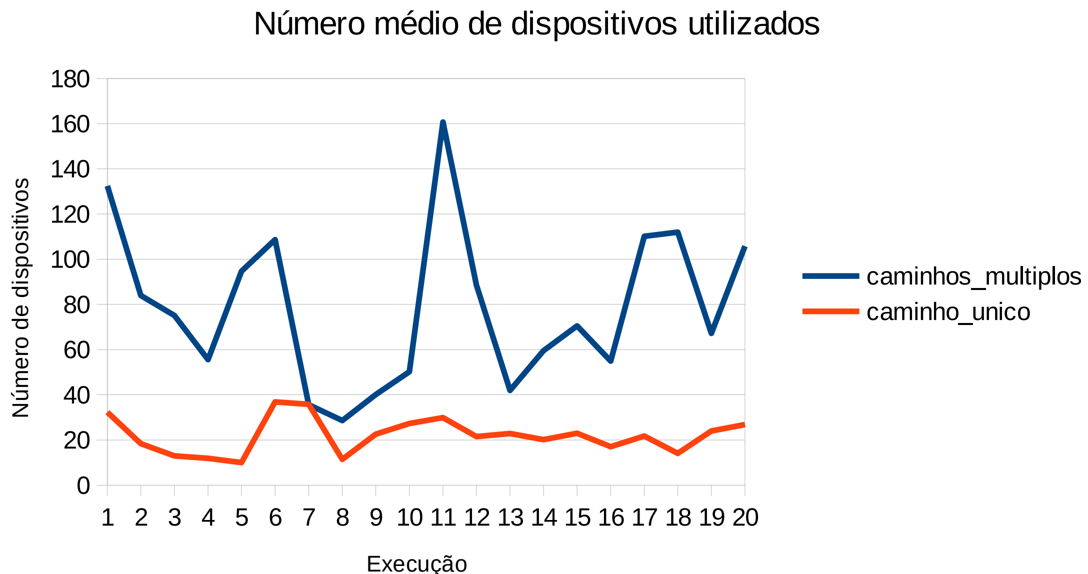

Figura 21 - Gráfico do número de dispositivos utilizados

Fonte: Elaborado pelo autor, 2019

Nas Figuras 21 e 22, é exibido a quantidade de dispositivos utilizado por ambos métodos, conforme é mostrado no gráfico. Através deles, é possível ver a diferença de utilização dos dispositivos por parte dos algoritmos, em uma rede contendo 700 dispositivos podendo ser utilizados, o algoritmo de caminho único não utiliza 50. Já o algoritmo de caminhos múltiplos, conseguiu diante dessa amostra alcançar uma quantia

de 160 dispositivos com disponibilidade de trafegar informação de uma origem até um destino pré definido. Diante disso, vale ressaltar o aproveitamento da rede por parte do método de caminhos múltiplos.

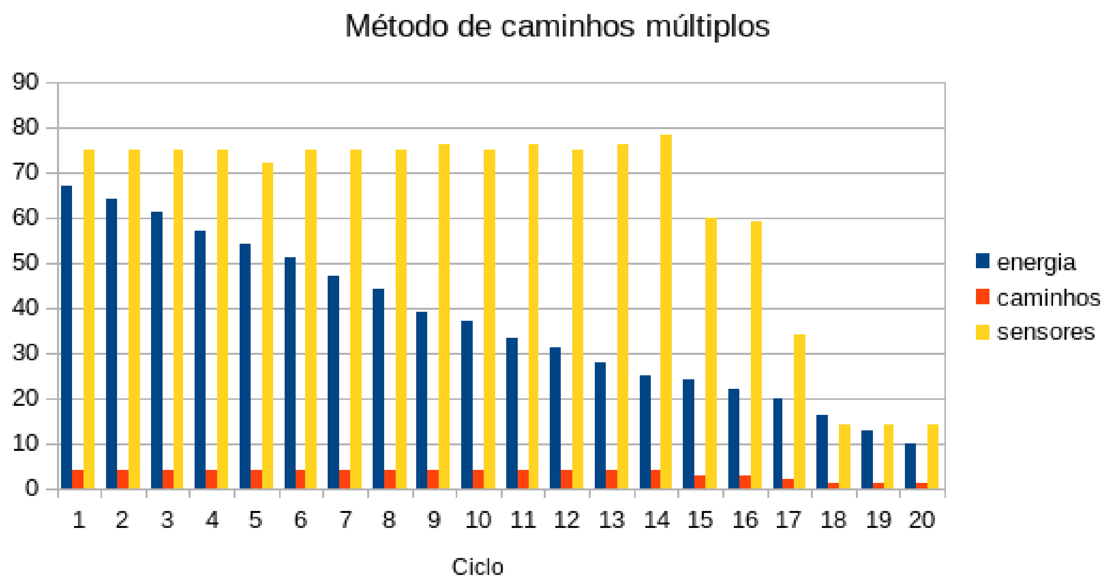

Figura 22 - Gráfico de dados da execução de maior ciclo 1

Fonte: Elaborado pelo autor, 2019

Na Figura 23 é apresentado o gráfico com informações da execução de número 37. Execução que marcou o pico de máximo da variável ciclo dentro do método de caminhos múltiplos, sendo possível observar o aproveitamento de uma quantia entre 70 e 80 sensores, mais do que 10% da rede total, durante a maior parte do tempo de roteamento. Também é visto a diminuição da energia da rede de forma gradual, mostrando que o método sempre busca rotas que aparentam no momento serem as melhores no que diz respeito a energia. O número de caminhos para essa execução, na maior parte do tempo permaneceu a mesma (4), o que favorece no momento de enviar

informações, mostrando que mesmo com a diminuição da bateria, o método consegue encontrar mesma quantidade de rotas por um longo período.

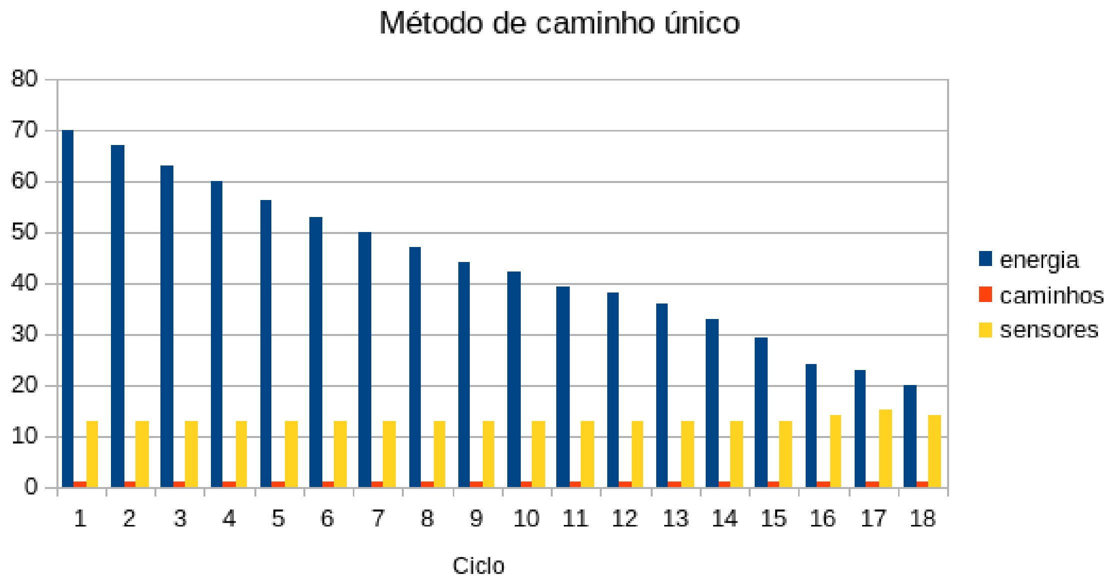

Figura 23 - Gráfico de dados da execução 37

Fonte: Elaborado pelo autor, 2019

Já a Figura 24 exibe o gráfico também da execução de número 37, porém esses dados são do método de caminho único. Diante do gráfico é possível observar a pequena utilização dos sensores por parte do algoritmo, em relação ao método de caminhos múltiplos, mas que a diminuição da carga possui o mesmo comportamento, de diminuição gradual, sendo essa uma desvantagem, já que diante de uma grande quantidade de sensores, o método não conseguiu alcançar a mesma quantidade de ciclos em uma mesma topologia, submetido as mesmas condições e buscando apenas um único caminho.

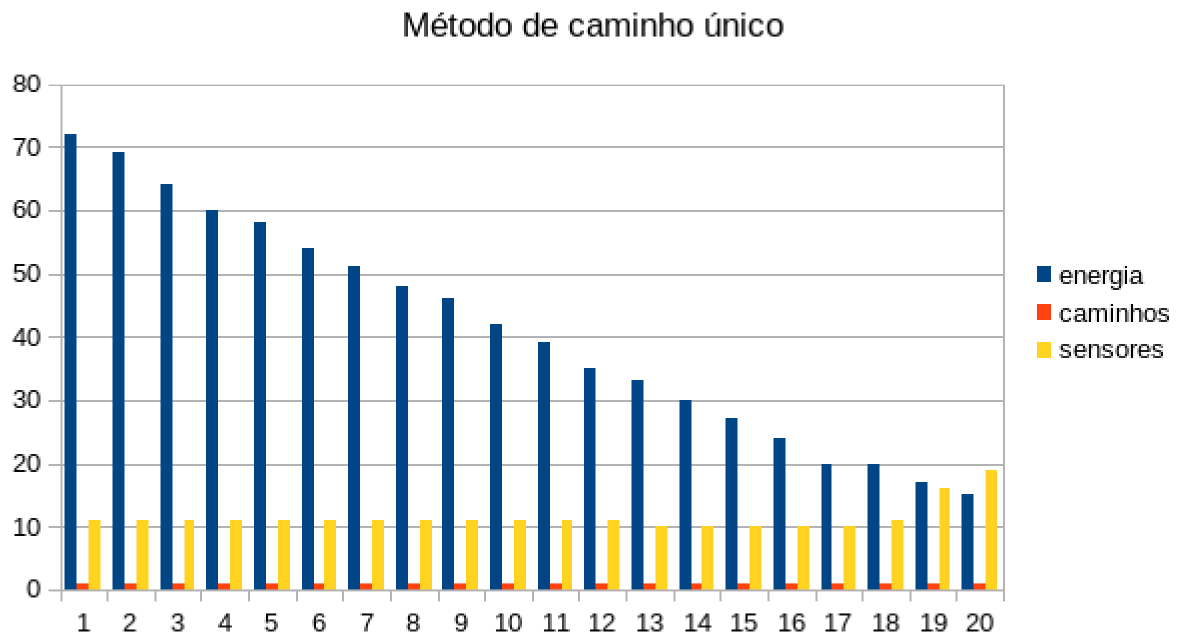

Figura 24 - Gráfico de dados da execução de maior pico 2

Fonte: Elaborado pelo autor, 2019

A Figura 25 apresenta os dados da execução 8, em que o método de caminho único conseguiu alcançar o pico de ciclos, um total de 20, mesma quantidade que o algoritmo de caminhos múltiplos, demonstrando as mesmas características que a Figura 24.

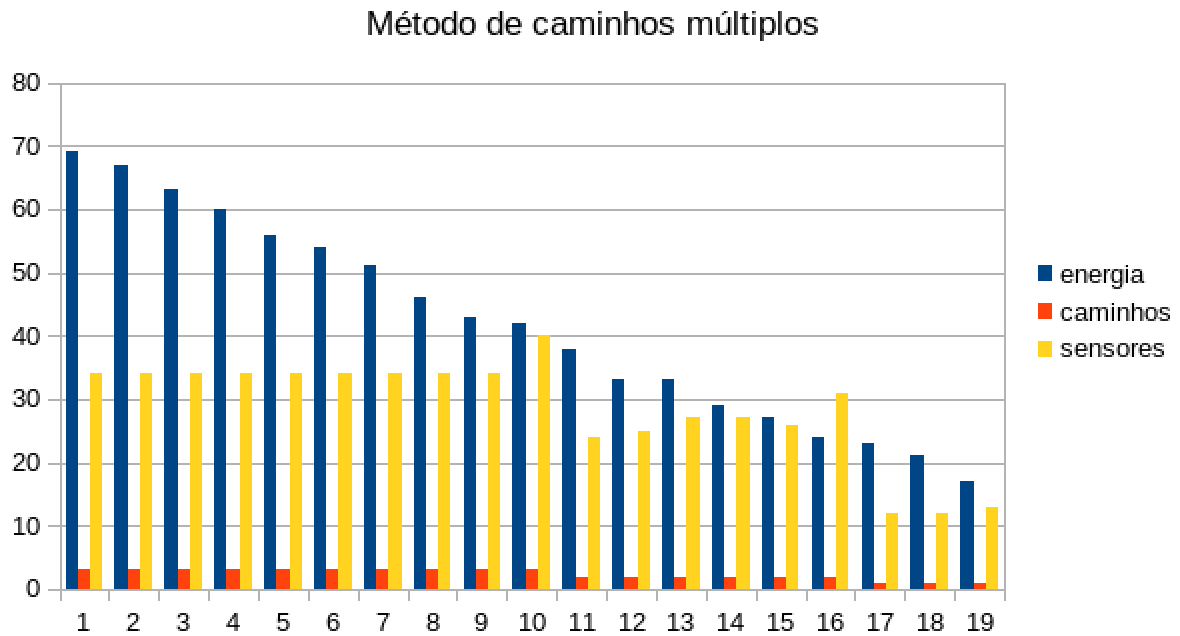

Figura 25 - Gráfico de dados da execução 8

Fonte: Elaborado pelo autor, 2019

A Figura 26 representa os dados da execução 8 para o método de caminhos múltiplos, que submetido as mesmas condições ambientais em uma mesma topologia que o algoritmo de caminho único conseguiu atingir o seu pico de ciclos, mostra que são números próximos, sendo 20 contra 19. Diante disso, mesmo o método de caminho único conseguindo uma maior quantidade de ciclos nesse momento, o de caminhos múltiplos consegue se aproximar, com as mesmas vantagens descritas na Figura 23.

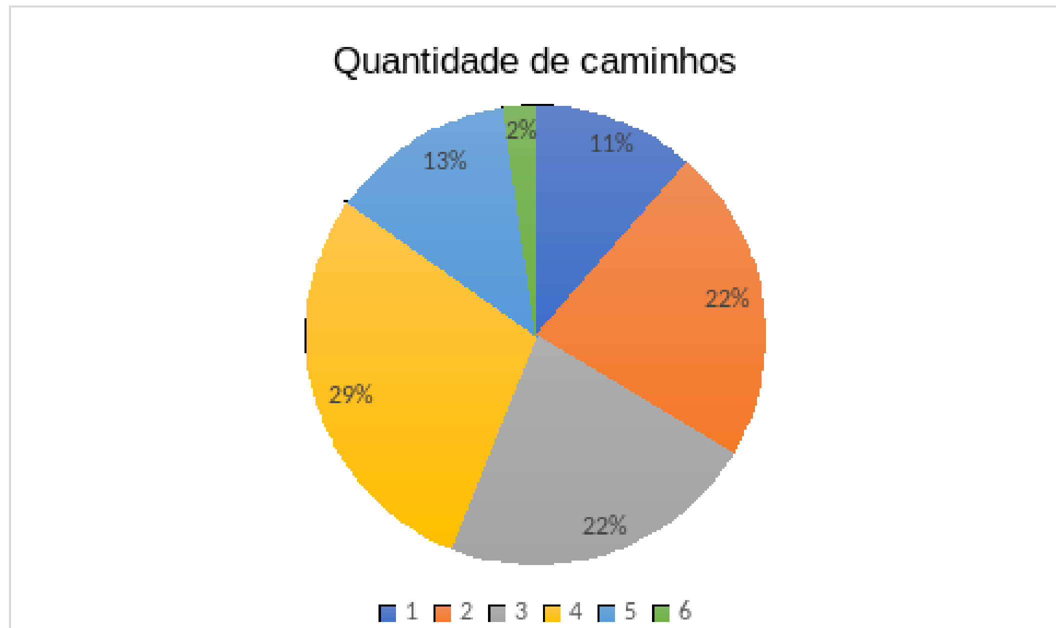

Figura 26 - Gráfico da quantidade de caminhos encontrados

Fonte: Elaborado pelo autor, 2019

Na Figura 27, é apresentado os dados referente a quantidade de caminhos encontrados pelo método de caminhos múltiplos, onde em maioria é possível encontra 2, 3 e até mesmo 4 rotas, com índices de frequências iguais a 22% para 2 e 3, e 29% para 4 rotas, durante a aplicação do método, que tem máximo de 6 rotas com uma baixa frequência. O número máximo de rotas encontrados é uma variável que depende da topologia e também da quantidade de sensores gerados, e para essas simulações foram utilizados 700.

A Tabela 1 abaixo exibe as médias finais dos resultados de cada variável observada.

Protocolo	Tempo	Caminhos	Dispositivos %	Energia %	Ciclos
multicaminhos	3,37	3,17	11	41,1	15,91
caminho único	3,36	1	3,14	42,2	15,82

Tabela 1 – Dados coletados

Fonte: Elaborado pelo autor, 2019

Mediante a coleta e representação dos dados, foi possível observar o melhor aproveitamento da rede por parte do algoritmo de caminhos múltiplos, que consegue buscar não apenas uma única rota, o que reflete na maior quantidade de dispositivos utilizados, não correndo o risco de particionar a rede sobrecarregando um único caminho. Em relação a energia dos sensores, tempo de execução e número de ciclos, os métodos desempenharam papéis semelhantes, com baixas variações para uma quantidade de setecentos sensores. A Tabela 1 evidencia essas interpretações.

6. CONCLUSÃO

Com o aprimoramento dos componentes que estruturam os sensores sem fio, e o desenvolvimento de câmeras de vigilância com baixo consumo de energia, permitiu expandir as possibilidades de implantação das redes *wireless*. Entretanto, a transmissão de dados visuais é mais custosa, pois necessita de maior largura de banda e processamento o que acarreta no aumento do consumo energético (CERQUEIRA, 2019). Utilizar os sensores com racionalidade, permitindo que a rede tenha uma maior durabilidade é um desafio, mas os estudos tendem a se intensificar, já que o campo tende a expandir em diversos ambientes.

Neste trabalho foi apresentado um estudo e implementação de dois métodos de roteamento baseado no algoritmo de Dijkstra, utilizando como métrica a eficiência energética, buscando através do método comparativo, otimizar o menor caminho que o algoritmo padrão encontra.

A utilização de vários modelos de gráficos apresentados na sessão 4, possibilitou uma melhor visualização dos resultados obtidos, podendo concluir através deles que ambos os algoritmos apresentaram resultados semelhantes, no que se diz respeito a energia, ciclos, e tempo de execução do algoritmo de Dijkstra. Sendo esses resultados satisfatórios, visto que um dos algoritmos tende a sobrecarregar os dispositivos de uma única rota (método de caminho único), e o outro tende a distribuir essa sobrecarga (método de multicaminhos).

Desenvolver a plotagem de gráficos, e simulação de dispositivos, em tempo de execução foi um desafio durante a fase de implementação do projeto, bem como a definição das variáveis de observação para realizar a comparação entre os dois métodos, porém conforme a evolução do projeto, foi possível vencer esses desafios.
Pode-se mencionar como pontos positivos desta análise, o desempenho do algoritmo de multicaminhos, que consegue distribuir o tráfego de rede, com características semelhantes ao método de caminho único.

Apontar um método de roteamento que faça uso racional dos componentes das redes de sensores sem fio, ainda é um desafio, porém a cada estudo realizado dentro

desse ambiente, uma nova contribuição é feita, para essa questão, auxiliando desde os estudos complexos aos mais simples.

Como proposta de trabalhos futuros tem-se a utilização de algoritmos de vetores de distâncias, juntamente com o algoritmo Bellman Ford para o mesmo problema. Outra ideia é formular uma nova métrica e utiliza-la no método implementado neste estudo, e posteriormente no algoritmo do Bellman Ford. E a implementação de uma função de transmissão, que permita utilizar qualquer um dos métodos de roteamento deste trabalho, e de trabalhos futuros, em um ambiente real. O objetivo desses trabalhos é o de comparar diferentes métodos em ambientes de simulações e em ambientes reais, fazendo inferências sobre os resultados e discuti-los.

REFERÊNCIAS BIBLIOGRÁFICAS

ARORA, Yash; PANDE, Himangi. **Energy Saving Multipath Routing Protocol for Wireless Sensor Networks.** India 2013. Disponível em: https://www.ijera.com/papers/Vol3_issue5/AA35152156.pdf Acesso em 10/04/2019 às 15:30

CERQUEIRA, M.V.B.; COSTA, D.G.. Um Modelo Matemático para Estimativas do Consumo de Energia em Redes de Sensores Visuais sem Fio. **TEMA (São Carlos)**, São Carlos , v. 20, n. 2, p. 257-276, Ago. 2019. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2179-84512019000200257&lng=en&nrm=iso>. Acesso em 23/09/2019 às 20:32 horas.

CHEN, Yunfeng; NASSER, Nidal. Energy-Balancing Multipath Routing Protocol for Wireless Sensor Networks. **Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks (QShine '06)**. Canada: ACM Digital Library, 2006. Disponível em: http://dx.doi.org/10.1145/1185373.1185401 Acesso em 04/04/2019 às 12:46 horas.

CORMEN, Thomas H.; LEISERSON, Charles Eric; RIVEST, Ronald Linn.; STEIN Clifford. **Algoritmos: Teoria e Prática**. Tradução de Arlete Simille Marques. 3. ed. Rio de Janeiro: Elsevier Editora LTDA, 2012.

GARAY, Jorge Rodolfo Beingolea. **Análise de desempenho de uma rede de sensores sem fio baseado no padrão ZIGBEE/IEEE 802.15.4**. São Paulo, 2007. Disponível em:

http://www.teses.usp.br/teses/disponiveis/3/3142/tde-08012008-113638/es.php Acesso em 14/03/2019 às 10:44 horas.

KUROSE, James F.; ROSS, Keith W. **Redes de computadores e a internet - uma abordagem top-down.** Tradução de Daniel Vieira. 6. ed. São Paulo: Pearson Education, 2014. Disponível em: https://www.passeidireto.com/arquivo/47886404/docgo-net-redes-de-computadores-e-a-internet-uma-abordagem-top-down-6-edicao-oficial-pdf Acesso em 25/03/2019 às 10:50 horas.

SHA, Kewei; GEHLOT, Jegnesh; GREVE, Robert. **Multipath Routing Techniques In Wireless Sensor Networks: A Survey.** Springer US, 2013. Disponível em: https://doi.org/10.1007/s11277-012-0723-2 Acesso em 07/10/2019 às 10:34 horas.

GOPI, Priya. **Multipath Routing in Wireless Sensor Networks: A Survey and Analysis**. IOSR Journal of Computer Engineering, 2014. Disponível em: https://www.researchgate.net/publication/269751012_Multipath_Routing_in_Wireless_Sensor_Networks_A_Survey_and_Analysis Acesso em 07/10/2019 às 15:09 horas.

CARVALHO, Marco Antonio G. Introdução à computação gráfica com OpenGL. **Ft.unicamp,** 2006. Disponível em: https://www.ft.unicamp.br/~magic/opengl/conceitos-iniciais.html Acesso em 08/10/2019 às 21:41 horas.

BEN-OTHMAN, Jalel; YAHYA, Bashir. Energy efficient and QoS based routing protocol for wireless sensor networks. **Journal of Parallel and Distributed Computing,** 2. VOL. Journal of Parallel and Distributed Computing, 2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731510000341> Acesso às 13:54 horas.

LIMA, Moysés Mendes de. **Evitando regiões de buraco de roteamento em redes de sensores sem fio.** 2019. 117 f. Tese (Doutorado em Informática) - Universidade Federal do Amazonas, Manaus, 2019. Disponível em: <https://tede.ufam.edu.br/handle/tede/6983> Acesso às 09:34 horas.

GNUPLOT, **gnuplot homepage,** 2019. Disponível em: <http://www.gnuplot.info/> Acesso às 20:28 horas.

APÊNDICE A – Algoritmo de Dijkstra alterado

O algoritmo elaborado no presente estudo tem como base o algoritmo de Dijskstra, em que o grafo continua sendo ponderado, porém os vértices serão entendidos como nós ou sensores sem fio, portando uma quantidade Pi de bateria inicial, ou seja, a quantidade de bateria máxima que o dispositivo tinha ao ser inserido na rede, e uma quantidade Pa de bateria atual. Assim será utilizado não somente os pesos das arestas do grafo, para encontrar o menor caminho mais energético, mas também levar em consideração a percentual energético dos dispositivos, formando com esses três argumentos, uma métrica. Segue abaixo a demonstração do funcionamento do algoritmo.

```
SPMEP(G, s)
1 INITIALIZE-SINGLE-SOURCE(G,s)
2 S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5        v ← extrai_min_metrica(Q)
6        for cada vértice p ∈ Adj[v]
7              do RELAX(v, p,metrica(v,p))
8        S ← S ∪ v
```

O algoritmo recebe como entrada, um grafo (G) conexo e ponderado e o nó inicial (s). A primeira linha, inicializa o grafo com 0 no vértice inicial s, e atribui infinito para os demais vértices. Já na linha 2 do algoritmo a estrutura S é inicializada com vazio. S é uma lista, e ela terá os nós que já foram visitados. A linha 3, Q também é uma lista, porém uma lista que contém todos os vértices/dispositivos do grafo. Já na linha 4, inicia-se uma iteração, que diz o seguinte, enquanto não for extraído todos os

vértices dentro de G, o algoritmo não finaliza. O laço é executado até a linha 9, em que na sequência, a linha 5, atribui-se a variável **v** o vértice que tem o menor acumulo de métricas até o momento (inicialmente o menor é o vértice inicial). Na linha 6 executa uma nova iteração para todos os vizinhos **p** de **v**, verificando na linha 7, se a métrica atual de **v** somado com a métrica de **v** até **p**, é menor do que o resultado acumulado até agora no vértice **p**, em caso afirmativo, **p** recebe essa soma. Na linha 8, o vértice **v** é armazenado no conjunto de vértices visitados, para que não seja visitado novamente. Esse processo se repete até que todos os vértices do grafo sejam visitados.

APÊNDICE B – Módulos do arquivo created_init.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>
#include <assert.h>
#include <unistd.h>
#include <string.h>

const double raio = 40;
//DIMENSÃOO DE BUSCA
const int MAXX = 600;
const int MAXY = 400;

const int LARGURA_TELA = 1400;
const int ALTURA_TELA = 800;

typedef struct sensor
{
   int Id;/*id = <posicao_vetor>*/
   int Pwi; /*Energia inicial do sensor. Gerada entre 70 e 99 */
   int Pwa; /* Energia atual, Gerada entre Pwi e (Pwi - 30)*/

   /*dois auxiliares para recuperar o valor original/inicial de pwi e pwa */
   int auxPwi;
   int auxPwa;

   /*x e y representa a posicao do sensor no plano */
   int x;
   int y;

   float **mtAdjMet; /*matriz [2][N], onde na primeira linha temos os sensores
adjacentes com
   sua respectiva distancia ao sensor referencia. E na segunda linha temos as metricas
para esses sensores.*/

   float **mtDijks; /*representa a matriz gerada quando o algoritmo de dijkstra eh
executado */
```

```
   int color; /*Representa as cores dos dispositivos na parte grafica
             0 - preta      - dispositivo qualquer   1 - verde      - representa o inicial
             2 - azul       - representa o final
             3 - rosado     - participa da comunicacao fim a fim
             4 - vermelho    - dispositivo sem carga
             */

   int sendData; /* Essa variavel auxilia no descobrimento de multicaminhos
                1 - se enviou dados
                 0 - caso nao envou*/

   int countRoutes; /*representa a quantidade de rotas encontradas */


}NODE;

/*AUXILIA O ALGORITMO DE DIJKSTRA*/
typedef struct queue{
   int id;
   float d;
   struct queue *prox;
}ENCADEADA;

//======================================== ALOCACAO DE NOHS, E
MATRIZES ========================================

NODE* alocaVetorNode (int N)
{
   NODE *V;

   V = malloc(sizeof(NODE) * N);
   return V;
}

int** alocaMatriz(int N)
{
   int i;
   int **M;

   M = malloc (sizeof(int *) * N);

   for( i = 0; i < N; i++)
   {
      M[i] = malloc (sizeof(int) * N);
   }
```

```
   return M;
}

float** alocaMatrizf(int N, int K)
{
   int i;
   float **M;

   M = malloc (sizeof(float *) * N);

   for( i = 0; i < N; i++)
   {
      M[i] = malloc (sizeof(float) * K);
   }
   return M;
}

//======================================== FIM DAS ALOCA��ES
======================================


//======================================== INICIALIZA��ES DOS NOHS E
MATRIZES ======================================
//objetivo: inicializar a matriz toda com 0 e para cada n� Ni, atribuir o id i+1
//inicializa as adjac�ncias, o contador de passos at� o �ltimo n�, o ID e a bateria
void inicializaMatriz (int N, int** Matriz)
{
   int i, j;
   for (i = 0; i < N; i++)
   {
      for(j = 0; j < N; j++)
      {
         Matriz[i][j] = 0;
      }
   }
}

//objetivo: inicializar a matriz toda com 0 e para cada n� Ni, atribuir o id i+1
//inicializa as adjac�ncias, o contador de passos at� o �ltimo n�, o ID e a bateria
void inicializaMatrizf (int N, int m, float valor, float** Matriz)
{
   int i, j;
   for (i = 0; i < N; i++)
   {
      for(j = 0; j < m; j++)
```

```
        {
            Matriz[i][j] = valor;
        }
    }
}

void inicializaNos(int N, NODE *Nos, int sink)
{
    int i;

    srand(time(NULL));
    for(i = 0; i < N; i++){
        Nos[i].Id = i;
        Nos[i].Pwi = rand() % 30 + 70;
        Nos[i].Pwa = rand() % 30 + (Nos[i].Pwi - 30);/*rand() % Nos[i].Pwi + 1 */
        Nos[i].mtAdjMet = alocaMatrizf(2,N);
        inicializaMatrizf(2, N, 0.0, Nos[i].mtAdjMet);
        Nos[i].mtDijks = alocaMatrizf(N,N+2);
        inicializaMatrizf(N, N+2, -1.0, Nos[i].mtDijks);
        Nos[i].color = 0;
        Nos[i].sendData = 0;
        Nos[i].countRoutes = 0;
        Nos[i].auxPwa = Nos[i].Pwa;
        Nos[i].auxPwi = Nos[i].Pwi;

    }
}

void inicializa_color(int N, NODE *nos){
    int i;

    for(i = 0; i < N; i++)
    {
        nos[i].color=0;
    }
```

APÊNDICE C – Módulos do arquivo router.h

```
#include "created_init.h"


//======================================= CONFIGURA A DISPOSI��O
DOS NOHS ======================================

float calcDist(int x, int y, int x1, int y1){
   float soma1, soma2, resultado;

   soma1 = x1 - x;
   soma1 = pow(soma1, 2.0);

   soma2 = y1 - y;
   soma2 = pow(soma2, 2.0);

   resultado = soma1+soma2;
   resultado = pow(resultado, 0.5);

   return resultado;
}

/*
OBJETIVO: VERIFICA SE AS COORDENADAS J� EST�O SENDO USADA,
EM CASO AFIRMATIVO PROCURA UMA COORDENADA V�LIDA E RETORNA
*/
void verificaCoordenadas (NODE *conjNos, NODE *No, int dimX, int dimY, int N)
{
   int i;
   float distancia = 0;
   int conectado = 0;/*supomos que a rede estah inicialmente conexa*/
   int xmax;
   int ymax;
        int cont = 0;

        /*enquanto o grafo nao for conexo, e os dispositivos nao estiverem a uma
distancia de pelo menos 10px de cada um nao sai do laco*/
        if(N > 0){
```

```
        while(!conectado){
                i = 0;
                while(i < N){
                    if (No->x == conjNos[i].x && No->y == conjNos[i].y)
                    {
                        i = 0;
                /*gera numeros entre xmin - xmax*/
                        No->x = (rand() % (dimX - 10)) + 5;
                        No->y = (rand() % (dimY - 10)) + 5;
                    }else
                    {
                        distancia = calcDist(conjNos[i].x, conjNos[i].y, No->x, No-
>y);

                        if(distancia <= raio && distancia >= 20){/* 10 <= distancia
<= 30*/

                            /*esta conectado. agora so preciso verificar se estah a
uma distancia de 10*/

                            i = N;
                            conectado = 1;
                        }
                        else{
                            i++;
                        }
                    }
                }

                if(conectado){
                    conectado = 0;
                    i = 0;
                    while(i < N && !conectado){
                        distancia = calcDist(conjNos[i].x, conjNos[i].y, No->x,
No->y);

                        if(distancia < 20.0){
                    No->x = (rand() % (dimX - 10)) + 5;
                                No->y = (rand() % (dimY - 10)) + 5;
                                conectado = 1;
                        }else{
                            i++;
                        }
                    }

                    if(!conectado){
                        conectado = 1;
                    }
                    else{
```

```
                        No->x = (rand() % (dimX - 10)) + 5;
                                No->y = (rand() % (dimY - 10)) + 5;
                                conectado = 0;
                            }
                    }else{
                            No->x = (rand() % (dimX - 10)) + 5;
                            No->y = (rand() % (dimY - 10)) + 5;
                    }
                }
        }
}

/*
FUN��O: ESPALHAR OS NOHS EM UM PLANO 2D, COM DIMENSOES MAXIMAS
dimX E dimY
*/
void espalhaDisp (NODE *conjNos, int N, int dimX, int dimY){
   NODE no1;
   int i;

   for(i = 0; i < N; i++){
      no1.x = (rand() % (dimX - 10)) + 5; /*gera numeros entre xmin - xmax*/
      no1.y = (rand() % (dimY - 10)) + 5;
      verificaCoordenadas(conjNos, &no1, dimX, dimY, i);
      conjNos[i].x = no1.x;
      conjNos[i].y = no1.y;
   }
}
//======================================== FIM DAS DISPOSI��ES DOS
NOHS ==========================================

//======================================== CONFIGURA AS LIGA��ES
DOS NOHS ==========================================
void defineConexoes(NODE *conjNos, int pos, int N, float **matriz){
   float distancia;
   int i = 0;
   NODE *aux;

   for(i = 0; i < N; i++){
      if(i != pos){
         distancia = calcDist(conjNos[pos].x, conjNos[pos].y, conjNos[i].x, conjNos[i].y);
         if(distancia <= raio){
            matriz[pos][i] = distancia;
            matriz[i][pos] = distancia;
         }
```

```
        }
    }
}

void procuraConexoes (NODE *conjNos, int N, float **matriz){
    int i;

    for(i = 0; i < N; i++){
        defineConexoes(conjNos, i, N, matriz);
    }
}
//=========================================== FIM DAS CONFIGURA��ES DE LIGACOES  =======================================

/*=========================================== PREPARANDO OS N�S PARA REALIZAREM ROTEAMENTO ============================*/

/*OBJETIVO: FAZER COM QUE CADA N� CONHE�A SEUS VIZINHOS*/
void conheceAdjacente(int N, float **matriz, NODE *conjNos){
    int i, j;

    for(i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            conjNos[i].mtAdjMet[0][j] = matriz[i][j];
        }
    }
}

float calcula_metrica(NODE *no, int adj){

    float divisao, denominador, numerador;

    divisao = ((float)no->Pwa / (float)no->Pwi);
    denominador = pow(divisao,1.0/3.0);
    numerador = pow(no->mtAdjMet[0][adj], 2.0)*0.00025;//o raio est� com 50

    if(denominador > 0.0)
        return numerador/denominador;
    else
        return 0.0;
}

/*Objetivo: CONHECER A M�TRICA DE CADA N� VIZINHO*/
void conhece_metrica(int N, NODE *conjNos){
    int i, j;
```

```
    float metrica;

    for(i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            if(conjNos[i].mtAdjMet[0][j] > 0){
                metrica = calcula_metrica(&conjNos[j], i);
                conjNos[i].mtAdjMet[1][j] = metrica;
            }else{
                conjNos[i].mtAdjMet[1][j] = 100.0;
            }
        }
    }
}
//========================================= FIM DAS CONFIGURAÇÕES DE
LIGACOES  =====================================

/* ==================================== INICIA AS TABELAS DE
ROTEAMENTOS ========================================*/

//essa funcao verifica se o nó pertence a fila de visitados
//ele retorna 1 caso positivo, e 0 caso negativo
int pertenceVisitadosFila (ENCADEADA *Visitados, NODE *NO)
{
    ENCADEADA *run;

    //percorremos a lista de visitados para ver se não já foi veriricado
    run = Visitados;

    while(run!= NULL)
    {
        if(run->id == NO->Id)
        {
            return 1;
        }else{
            run = run->prox;
        }
    }
    return 0;
}

//objetivo: adicionar o dispositivo No(varíável) a lista de visitados
void AdcListaVisitados(ENCADEADA **visitados, ENCADEADA *No)
{
    ENCADEADA *run;
```

```
    run = *visitados;
    if(run != NULL){
        while(run->prox != NULL) run = run->prox;
        No->prox = NULL;
        run->prox = No;
    }
    else{
        *visitados = No;
    }


}

void se_existe_remove(ENCADEADA **pt, int id){

    ENCADEADA *del;

    if(*pt != NULL){
        if((*pt)->id == id){
            del = (*pt);
            *pt = del->prox;
            free(del);
        }else{
            se_existe_remove(&(*pt)->prox, id);
        }
    }
}

/*Fila de prioridade mínima
Está ordenada pela menor distância,
se houver distâncias iguais, o primeiro
dos iguais será o primeiro que entrou.

 OBJETIVO: CONTINUAR COM A PROPRIEDADE DA FILA, ADICIONANDO UM NÓ
 EM SUA POSIÇÃO, E CASO ESSE NÓ JÁ PERTENCE A FILA (POIS ADICIONAMOS
ELE NOVAMENTE
 QUANDO ENCONTRAMOS UM NOVO CAMINHO MENOR) TEMOS QUE DELETAR
O ANTIGO, ENTÃO ADICIONAMOS O
 NOVO, E REMOVEMOS O ANTIGO*/
void adc_fila(ENCADEADA **fila, int id, float d){
    ENCADEADA *novo;

    if(*fila == NULL){//se nao tiver nada na fila, eu aloco o primeiro nó
        assert(((*fila)=malloc(sizeof(ENCADEADA))) != NULL);
        if((*fila)!=NULL)/*seguranca*/
```

```
        {
            (*fila)->id=id;
            (*fila)->d=d;
            (*fila)->prox=NULL;
        }else{
            printf("Abort!\n");
            exit(1);
        }

    }else{
        if((*fila)->d <= d){
            adc_fila(&(*fila)->prox, id, d);
        }else{
            novo = malloc(sizeof(ENCADEADA));
            novo->id = id;
            novo->d = d;
            novo->prox = *fila;
            *fila = novo;

        }

    }
}

ENCADEADA* remove_min(ENCADEADA **fila){
    ENCADEADA *F;

    F = (*fila);
    (*fila) = (*fila)->prox;

    return F;
}

void inicia_d_p(float *d, int *p, int id, int N){
    int i;

    for(i = 0; i < N; i++){
        d[i] = 10000.0;
        p[i] = -1;
    }
    d[id] = 0.0;
}

void dijkstra(NODE *INI, NODE *conjNos, int N){
    ENCADEADA *FILA;
```

```
    ENCADEADA *u;
    ENCADEADA *visitados;
    ENCADEADA *ax;
    FILE *fp;

    int i, aux;
    float d[N];//representa a distancia acumulada, iniciada com 10000,
    //pois a probabilidade de ter essa distancia é muito baixa, pois é um número alto.
    // O inicial é 0;
    int p[N];//representa o predecessor, iniciada com -1, pois nenhum pai terá o id -1;
    float metrica;
    int pos_u, pos_pai;//obtem a posição de u e do pai

    inicia_d_p(d, p, INI->Id, N);/*inicializacao semelhante ao dijkstra padrao*/
    FILA = NULL;
    visitados = NULL;
    adc_fila(&FILA, INI->Id, d[INI->Id]);
    if(FILA == NULL){
       printf("Fila nao alocada!");
       exit(2);
    }

    /*if((fp = fopen("algoDijks", "a+")) == NULL)
    {
       puts("erro na abertura/criacao do arquivo algoDijks");
       exit(2);
    }*/

    //fprintf(fp, "Anahlise noh %d com predecessor = %d\n", pos_u, p[pos_u]);

    while(FILA != NULL){
       u = remove_min(&FILA); /*remove da fila o sensor de menor metrica */
       pos_u = u->id; /*salvo o identificador dele, que representa a sua posicao no vetor
*/
       for(i = 0; i < N; i++){
          if(conjNos[u->id].mtAdjMet[1][i] > 0.0 /*metrica maior que 0 sao vizinhos */ &&
conjNos[u->id].mtAdjMet[1][i] < 100.0 && !pertenceVisitadosFila(visitados, &conjNos[i])
&& conjNos[i].sendData == 0){
             metrica = conjNos[pos_u].mtAdjMet[1][i];

             /*fprintf(fp,"Noh = %d Pwa = %d Pwi = %d dist = %.3f\n", i, conjNos[i].Pwa,
conjNos[i].Pwi, conjNos[pos_u].mtAdjMet[0][i]);*/

             if(d[i] > d[pos_u] + metrica){
```

```
                if(p[i] != -1){//aqui significa que ele possuia um pai, e o seu pai deverá ser
corrigido
                    //pois uma distância menor foi encontrada
                    aux = p[i];
                    INI->mtDijks[aux][i] = 0.0;
                }
                if(u->id == INI->Id){//se for o primeiro que estamos analisando, o destino
são os próprios filhos.
                    //A posição N+1 é o destino do pacote
                    INI->mtDijks[i][N+1] = i;
                }else{//o destino do pacote será o mesmo destino do pai dele, ou seja, o nó
atual que está analisando
                    INI->mtDijks[i][N+1] = INI->mtDijks[pos_u][N+1];
                }
                INI->mtDijks[i][N] = u->id;//identifica o predecessor dele, (FAMOSO PI DE
U)

                //atualiza as variáveis internas da função
                d[i] = d[pos_u] + metrica;
                p[i] = pos_u;

                INI->mtDijks[pos_u][i] = d[i];//atualiza a matriz de u, adicionando a distancia
acumulada para o destino i

                se_existe_remove(&FILA, i);

                adc_fila(&FILA, i, d[i]);//adiciona i na fila, (i representa não só a posição
mas também é o identificador do nó)

            }
        }
    }
    AdcListaVisitados(&visitados, u);
  }
  //fprintf(fp,"=====================================================\n");

//    fclose(fp)
}
```

APÊNDICE D – Módulo principal do programa main.c

```
#include "router.h"
#include "../src/gnuplot_i.h"
#include <pthread.h>
#include <GL/glut.h>

#define PI 3.14159265358979323846

/*lista para obter as rotas encontradas, utilizando uma matriz
2xN, na qual a primeira linha representa os id's/ip's dos
 dispositivos, e a segunda linha representa as baterias dos dispositivos*/
typedef struct routes{
   double id;
   double energy;
   int fim; /*1 para fim do primeiro caminho, 0 para dispositivo intermediario
         assim podemos fazer graficos limitados, definir quando parar de procurar,
         ou pode ser desnecessario em alguns momentos. */
   struct routes * prox;
} ROTAS;

NODE *Nos;
int N;/*representa a quantidade de dispositivos */
int dispMenu = -1;/*representa o dispositivo que foi clicado */
int inicial = -1, final = -1;/*representa o dispositivo inicial e final */
int ax = LARGURA_TELA - 10, ay = ALTURA_TELA - 10; /*representa os eixos x e y do
plano do opengl. Ele nao eh o tamanho da janela */
int click = -1;/*representa o dispositivo eu devo atribuir a cor de inicial ou final
         se -1 -> nao foi atribuido o incial
         se 0 -> ja foi atribuido o inicial, resta o dispositivo final
         se 1 -> os dois dispositivos ja foram definidos e pode iniciar o roteamento */

/*médias  */
double med_temp_dijks = 0.0;
int qt_temp_dijks = 0;

int med_energia = 0; /*media das energias utilizadas no caminho */
int med_dispositivo = 0;/*media da quantidade de dispositivo utilizado no caminho */
```

```
int med_caminhos = 0;
int num_ciclos = 0; /*representa o numero de vezes que os caminhos poderam ser
encontrados com sucesso */

typedef struct dados_thread{double *x;
   double *y;
   int cont;
}DATE_THREAD;

typedef struct data_rede{
   NODE *Nos;
   int N;
}Data;

void Desenha(void);
void organiza();

//====================================== EXIBE O CONTE�DO DAS
MATRIZES ===================================
//mostra uma matriz de adjac�ncia
void mostra (int N, int** matriz)
{
   int i, j;

   printf("  ");
   for (int i = 0; i < N; ++i)
   {
      printf("%i ",i);
   }
   printf("\n------------------------------\n");
   for (i = 0; i < N; i++)
   {
      printf("%i|",i );
      for(j = 0; j < N; j++)
      {
         printf("%d ", matriz[i][j]);
      }
      printf("\n");
   }
   printf("------------------------------\n");
}

//mostra uma matriz de adjacencia
void mostraf (int N, float** matriz)
{
```

```
    int i, j;

    printf("  ");
    for (int i = 0; i < N; ++i)
    {
        printf("%i     ",i);
    }
    printf("\n-----------------------------\n");
    for (i = 0; i < N; i++)
    {
        printf("%i|",i );
        for(j = 0; j < N; j++)
        {
            printf("%.2f ", matriz[i][j]);
        }
        printf("\n");
    }
    printf("-----------------------------\n");
}
//======================================= FIM DAS EXIBICOES
======================================

/*=======================================GRAVACAO DA MEDIA DE
BATERIAS E NUMERO DE NÓS=================================== */
void grava_energia()
{
    FILE *fp;
    int qtbarra_ene;
    char arq;
    double media_energ = 0.0;
    double media_dispos = 0.0;
    double media_caminhos = 0.0;


    if(num_ciclos){
        media_dispos = med_dispositivo/num_ciclos;
        media_caminhos = med_caminhos/num_ciclos;
    }
    else{
        puts("num_ciclo == 0");
    }

    if(med_dispositivo)
        media_energ = med_energia/med_dispositivo;
    else
```

```
      puts("med_dispositivo == 0");

   if((fp = fopen("./dados/media_energia_dispositivo.txt", "a+")) == NULL)
   {
      puts("Erro na abertura do arquivo de log para construcao do grafico de tempo dijk!\
n");
      exit(1);
   }

   rewind(fp);

   qtbarra_ene = 0;
   while((arq = getc(fp)) != EOF)
   {
      if(arq == '\n')
         qtbarra_ene++;
   }
   qtbarra_ene++;

   fprintf(fp,"%.6f %.6f %.6f %d\n",media_caminhos, media_dispos, media_energ,
num_ciclos);

   fclose(fp);
}

/*=====================================GRAVACAO DA MEDIA DE
TEMPO DE EXECUCAO PARA O ALGORITMO DE
DIJKSTRA================================== */

void grava_medias_dijkstra()
{
   FILE *fp;
   int qtbarra_ene;
   char arq;
   double media;

   if(qt_temp_dijks)
      media = med_temp_dijks/qt_temp_dijks;
   else
      puts("zerooo!");


   if((fp = fopen("./dados/media_exec_dijks.txt", "a+")) == NULL)
   {
```

```
        puts("Erro na abertura do arquivo de log para construcao do grafico de tempo dijk!\
n");
        exit(1);
    }

    rewind(fp);

    qtbarra_ene = 0;
    while((arq = getc(fp)) != EOF)
    {
        if(arq == '\n')
            qtbarra_ene++;
    }
    qtbarra_ene++;

    fprintf(fp,"%.6f\n", media);

    fclose(fp);

    grava_energia();
}

void teste(ROTAS *p){
    puts("encontrou rota\n");

    while(p != NULL){
        printf("(%.3f , %.3f) ",p->id, p->energy);
        if(p->fim && p->prox != NULL)
            puts("outro caminho\n");
        p=p->prox;
    }
}

void grava_cada_energia_caminho_dispositivo(int baterias, int nDispositivos, int
caminhos, char *nome_arq){
    FILE *fp;
    float media= 0.0;

    if((fp = fopen(nome_arq,"a+")) == NULL){
        puts("Erro na abertura do arquivo energia_multipath");
        exit(1);
    }

    if(nDispositivos == 0){
        puts("Nao foi possivel gravar");
```

```
    }else{
        media = baterias/nDispositivos;

        fprintf(fp,"%.6f %d %d\n",media,caminhos, nDispositivos);
    }

    fclose(fp);

}

void finaliza_arquivo(char *nome_arq){
    FILE *fp;

    if((fp = fopen(nome_arq,"a+")) == NULL){
        puts("Erro na abertura do arquivo finaliza_arquivoenergia_ciclo");
        exit(1);
    }

    fprintf(fp,"=====================FIM====================\n");

    fclose(fp);

}

void grava_tempo_multipath(double inicio, double fim, char *nome){
    FILE *fp;

    if((fp = fopen(nome,"a+")) == NULL){
        puts("Erro na abertura do arquivo tempo_multipath");
        exit(1);
    }

    fprintf(fp, "%.6f\n",(fim-inicio));

    fclose(fp);
}

void grava_ciclos(int quantidade, char *nome){
    FILE *fp;

    if((fp = fopen(nome,"a+")) == NULL){
        puts("Erro na abertura do arquivo ciclo");
        exit(1);
    }
```

```
    fprintf(fp, "%d\n",quantidade);

    fclose(fp);
}

void *print_time(void *args){

    struct timeval time_inicial, time_final;
    double tIni, tFim;
    int i, j, aux;
    int cargaRetirada; /*potencia de carga que será retirado da bateria residual*/
    gnuplot_ctrl **graph; /*para funcionar o grafico, basta descomentar tudo que tem
relação com graph */
    char msg[100]; /*msg para enviar no plot*/
    FILE *fp;
    char arq;
    int qtbarra_ene;
    int num_arq;
    char nome_arq[20];
    ROTAS *rotas; /*obtem os caminhos encontrados */
    ROTAS *auxRotas;
    /*variaveis para contar a quantidade de caminhos e auxiliar
    na plotagem do grafico */
    int total_caminhos = 0;
    int conta_dispositivos;
    int multicaminhos = 1; /*identifica se o algoritmo a ser executado é o multicaminhos
ou o de caminho unico, incialmente
                        eh o algoritmo de caminhos multiplos */
    int controla_qt_caminhos = 0; /*ele controla a quantidade de caminhos para limitar
para apenas um caminho quando nao for o
                        o algoritmo de caminhos multiplos que estiver executando,
inicialmente nao necessita de controle
                        pois o primeiro algoritmo a iniciar eh o de caminhos multiplos */
    int media_energia_local;
    int FUNCIONA = 1;


    printf("Resposta para rota em dijkstra:::::\n");

    if(Nos[inicial].mtDijks[final][N+1] < 0.0){//o N+1 representa o proximo dispositivo, se
ele é -1 significa que nao há rotas até o proximo nó -> rede desconexa
        printf("Falha na comunicação com o dispositivo de id = %i",final);
        Nos[inicial].color = 4;
        grava_medias_dijkstra();
    }
```

```
num_arq = 0;
rotas = NULL; /*como nao foi encontrado nenhum caminho, inicializamos a variavel */

while(FUNCIONA) {

   // to refresh the window it calls display() function
   glutPostRedisplay();

   gettimeofday(&time_inicial, NULL);
   tIni = (double) time_inicial.tv_usec / 1000000 + (double) time_inicial.tv_sec;

   tFim = 0;
   while (tFim - tIni < 4.0) {
      gettimeofday(&time_final, NULL);
      tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;
   }

   multicaminhos = 1;

   click = 1;
   if(click == 1){
      inicializa_color(N, Nos);
      Nos[inicial].color = 1;
      Nos[final].color = 2;
      for (i = 0; i < N; i++) {
         cargaRetirada = rand() % 8;
         if(Nos[i].Pwa - cargaRetirada < 0){
            Nos[i].Pwa = 0;
            Nos[i].color = 4;
         }else{
            Nos[i].Pwa -= cargaRetirada;
            Nos[i].sendData = 0;
         }
      }
      conhece_metrica(N, Nos);/*Conhece a nova metrica dos dispositivos*/

      for(i = 0; i < N; i++){
         inicializaMatrizf(N, N+2, -1.0, Nos[i].mtDijks);
      }

      for(i = 0; i < total_caminhos; i++)
      {
         gnuplot_close(graph[i]);
      }
```

```
        if(total_caminhos != 0)
           free(graph);

        total_caminhos = 0;

        gettimeofday(&time_inicial, NULL);
        tIni = (double) time_inicial.tv_usec / 1000000 + (double) time_inicial.tv_sec;
        for (i = 0; i < N; i++) {
           if(Nos[i].Pwa > 0) /*executa o dijkstra somente para os sensores com
                          bateria positiva */
              dijkstra(&Nos[i], Nos, N);
        }
        gettimeofday(&time_final, NULL);
        tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;

        med_temp_dijks += (tFim - tIni); /*variavel global, para acumular o tempo da
execucao do algoritmo de dijkstra e calculo de rota */
        qt_temp_dijks++;/*variavel global para controlar quantas vezes foi rodado o
algoritmo e encontrado o caminho, consideramos como quantidade de ciclos tambem */

        auxRotas = NULL;
        rotas = NULL;
        while(Nos[inicial].mtDijks[final][N+1] != -1 && multicaminhos == 1){
           printf("Resposta para rota em dijkstra::::\n");
           total_caminhos++;
           aux = inicial;
           while (aux != final){
              if(Nos[aux].Pwa == 0){
                 printf("Dispositivo descarregado de id = %i\n", aux);
                 inicializa_color(N, Nos);
                 Nos[aux].color = 4;
                 aux = final;
                 grava_medias_dijkstra();
                 /*gravacao do arquivo de log do tempo de execucao do algoritmo de
dijks */
              }else{
                 if((int)Nos[aux].mtDijks[final][N+1] == -1){
                    teste(auxRotas);
                    if(Nos[aux].countRoutes == 0)
                       printf("Rede desconexa! %d\n",aux);
                    else
                       printf("Numero total de rotas = %d\n", Nos[aux].countRoutes);
                    aux = final;
                    //grava_medias_dijkstra();
```

```
                        } else{
                            if(rotas == NULL){
                                rotas = malloc(sizeof(ROTAS));
                                rotas->prox = NULL;
                                auxRotas = rotas;
                            }
                            else{
                                rotas->prox = malloc(sizeof(ROTAS));
                                rotas = rotas->prox;
                                rotas->prox = NULL;
                            }

                            rotas->id = (double)Nos[aux].Id; /*double pois eh o que o grafico pede
*/
                            rotas->energy = (double)Nos[aux].Pwa;
                            rotas->fim = 0;

                            med_energia += rotas->energy;/*acumula a energia para gerar uma
media para o arquivo */

                            /*se o dispositivo analisado eh o inicial, entao encontramos uma rota a
mais */
                            if(aux == inicial){
                                Nos[aux].countRoutes++;
                            }else{
                                Nos[aux].sendData = 1;
                                Nos[aux].color = 3;
                            }

                            aux = (int)Nos[aux].mtDijks[final][N+1];

                            if(aux == final){
                                rotas->prox = malloc(sizeof(ROTAS));
                                rotas = rotas->prox;
                                rotas->prox = NULL;

                                rotas->id = (double)Nos[aux].Id; /*double pois eh o que o grafico
pede */
                                rotas->energy = (double)Nos[aux].Pwa;
                                rotas->fim = 1;
                                med_energia += rotas->energy;
                                if(controla_qt_caminhos)
                                    multicaminhos = 0;
                            }
                        }
```

```
        }

    }

    conhece_metrica(N, Nos);/*Conhece a nova metrica dos dispositivos*/

    for(i = 0; i < N; i++){
        inicializaMatrizf(N, N+2, -1.0, Nos[i].mtDijks);
    }

    if(multicaminhos){ /*caso for o algoritmo de multicaminhos que esta
executando, procuramos mais caminhos e guardamos o tempo */
        grava_tempo_multipath(tIni, tFim, "tempo_multipath.txt");
        gettimeofday(&time_inicial, NULL);
        tIni = (double) time_inicial.tv_usec / 1000000 + (double) time_inicial.tv_sec;
        for (i = 0; i < N; i++) {
            if(Nos[i].Pwa > 0)
                dijkstra(&Nos[i], Nos, N);
        }

        gettimeofday(&time_final, NULL);
        tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;
    }

    med_temp_dijks += (tFim - tIni); /*variavel global, para acumular o tempo da
execucao do algoritmo de dijkstra e calculo de rota */
    qt_temp_dijks++;/*variavel global para controlar quantas vezes foi rodado o
algoritmo e encontrado o caminho, consideramos como quantidade de ciclos tambem */
}

if(total_caminhos == 0) {

    if(controla_qt_caminhos){//essa variavel inicia com 0, e so altera o seu valor
para 1 uma unica vez, depois so
                        //atribui a ela o mesmo valor 1, que significa que esta
executando o metodo de caminho unico
        FUNCIONA = 0;
        grava_ciclos(num_ciclos, "ciclos_singlepath.txt");
        finaliza_arquivo("ciclos_singlepath.txt");
        finaliza_arquivo("ciclos_multipath.txt");

    }
    else{
        finaliza_arquivo("energia_caminho_dispositivo_multipath.txt");
        grava_ciclos(num_ciclos, "ciclos_multipath.txt");
```

```
        }

        if(Nos[inicial].Pwa == 0) {
            puts("\nDispositivo inicial descarregado");
        } else {
            if (Nos[final].Pwa == 0)
                puts("\nDispositivo final descarregado");
            else
                puts("\nRede desconexa");
        }

        /*if(!controla_qt_caminhos){
            grava_medias_dijkstra();
        }else{
            /*grava medias para algoritmo de caminho unico
    }*/
        controla_qt_caminhos = 1;

        for(i = 0; i < N; i++){
            Nos[i].Pwa = Nos[i].auxPwa;
            Nos[i].Pwi = Nos[i].auxPwi;
        }
        num_ciclos = 0;
        med_temp_dijks = 0.0;
        qt_temp_dijks = 0;
        med_energia = 0;
        med_dispositivo = 0;
        med_caminhos = 0;
    }

    num_ciclos++;
    rotas = auxRotas;
    conta_dispositivos = 0;
    media_energia_local = 0;
    while(rotas != NULL){
        media_energia_local += rotas->energy;
        rotas = rotas->prox;
        conta_dispositivos++;
    }

    if(controla_qt_caminhos){
        if(total_caminhos > 0){
            grava_tempo_multipath(tIni, tFim, "tempo_singlepath.txt");
            grava_cada_energia_caminho_dispositivo(media_energia_local,
conta_dispositivos, total_caminhos, "energia_caminho_dispositivo_unico.txt");
```

```
        }
        else{
            finaliza_arquivo("energia_caminho_dispositivo_unico.txt");
        }

    }else{
        if(total_caminhos > 0){
            grava_cada_energia_caminho_dispositivo(media_energia_local,
conta_dispositivos, total_caminhos, "energia_caminho_dispositivo_multipath.txt");
            finaliza_arquivo("tempo_multipath.txt");
        }
    }

    med_caminhos += total_caminhos;

    med_dispositivo += conta_dispositivos;
    /*unica vez que somo as baterias do dispositivo inicial e do final eh aqui */
    med_energia += Nos[inicial].Pwa;
    med_energia += Nos[final].Pwa;

    rotas = auxRotas;

    //arquivo para mostrar o grafico de baterias comentado

    for(i = 0; i < total_caminhos; i++){

        sprintf(nome_arq,"histograma/histog_%d.txt",i);

        if((fp = fopen(nome_arq, "w")) == NULL)
        {
            puts("Erro na abertura do arquivo de log para constru do caminho!\n");
        }

        if(rotas != NULL){
            while(!rotas->fim){
                fprintf(fp, "%.1f %.1f\n",rotas->id, rotas->energy);
                rotas=rotas->prox;
            }

            fprintf(fp, "%.1f %.1f\n",rotas->id, rotas->energy);
            if(rotas->prox != NULL)
                rotas=rotas->prox;
        }

        fclose(fp);
```

```
        }

        if(total_caminhos > 0)
           graph = malloc(sizeof(gnuplot_ctrl*) * total_caminhos);

        for(i = 0; i < total_caminhos; i++)
           graph[i] = gnuplot_init();


        for(i = 0; i < total_caminhos; i++){
           sprintf(nome_arq,"histog_%d.txt",i);
           gnuplot_resetplot(graph[i]);
           if(auxRotas != NULL){
              gnuplot_cmd(graph[i], "set style data histograms");
              gnuplot_cmd(graph[i], "set yrange[0:100]");
              gnuplot_set_xlabel(graph[i], "ID node");
              gnuplot_set_ylabel(graph[i], "Energy");
              gnuplot_cmd(graph[i],"set key above right");
              sprintf(msg,"plot './histograma/histog_%d.txt' using 2:xtic(1) title 'Trafego de
%i para %i -> %s' ",i, inicial, final,nome_arq);
              gnuplot_cmd(graph[i], msg);


           }else{
              gnuplot_cmd(graph[i], "plot title 'no data'");
           }
        }

        click = -1;
     }

     rotas = auxRotas;
     while(rotas != NULL){
        auxRotas = rotas;
        rotas = rotas->prox;
        free(auxRotas);
     }
  }
  exit(0);
}

/*funcao para gravar a matriz de distancia do tipo float em um arquivo de nome igual a
file_name */
void file_rec_matrix(char *file_name, float **mt){
  char fname[20];
```

```
    FILE *fp;/*Arquivo para geração de logs*/
    int i, j;

    sprintf(fname,"%s.txt", file_name);
    if((fp = fopen(fname, "w")) == NULL)
    {
        puts("Erro na abertura do arquivo de log para construcao do caminho!\n");
        exit(1);
    }

    for(i = 0; i < N; i++){
        for(j = 0; j < N; j++){
            fprintf(fp, "%.3f ",mt[i][j]);
        }
        fprintf(fp,"\n");
    }
    fclose(fp);
}

/*funcao para gravar a matriz de dijkstra de cada dispositivo
do tipo float em um arquivo de nome igual a file_name */
void file_rec_matrix_dij(char *file_name){
    char fname[20];
    FILE *fp;/*Arquivo para geração de logs*/
    int i, j,l;

    sprintf(fname,"%s.txt", file_name);
    if((fp = fopen(fname, "w")) == NULL)
    {
        puts("Erro na abertura do arquivo de log para construcao do caminho!\n");
        exit(1);
    }

    for(l =0; l < N; l++){
        fprintf(fp, "Noh %d\n\n",l);
        for(i = 0; i < N; i++){
            for(j = 0; j < N; j++){
                fprintf(fp, "%.3f ",Nos[l].mtDijks[i][j]);
            }
            fprintf(fp,"\n");
        }
    }
    fclose(fp);
}
```

```
/*funcao para gravar a matriz de metricas do tipo float
 de cada dispositivo em um arquivo de nome igual a file_name */
void file_rec_matrix_met(char *file_name){
   char fname[20];
   FILE *fp;/*Arquivo para geração de logs*/
   int i, j,l;

   sprintf(fname,"%s.txt", file_name);
   if((fp = fopen(fname, "w")) == NULL)
   {
      puts("Erro na abertura do arquivo de log para construcao do caminho!\n");
      exit(1);
   }

   for(l =0; l < N; l++){
      fprintf(fp, "Noh %d\n\n",l);
      for(i = 0; i < 2; i++){
         for(j = 0; j < N; j++){
            fprintf(fp, "%.3f ",Nos[l].mtAdjMet[i][j]);
         }
         fprintf(fp,"\n");
      }
   }
   fclose(fp);
}

void init(void){

   /*Obtencao do tempo de execucao*/
   struct timeval time_inicial, time_final;
   double tIni, tFim;
   float **MatrizDistancia;
   int i, j, l;

   glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glEnable(GL_LINE_SMOOTH);
        glEnable(GL_POLYGON_SMOOTH);
   glEnable(GL_POINT_SMOOTH);

   Nos = alocaVetorNode(N);
   MatrizDistancia = alocaMatrizf(N, N);

   inicializaNos(N, Nos, N);
   inicializaMatrizf(N, N, 0.0, MatrizDistancia);
```

```
    gettimeofday(&time_inicial,NULL);
    espalhaDisp(Nos, N, ax, ay);
    gettimeofday(&time_final,NULL);

    gettimeofday(&time_inicial,NULL);
    procuraConexoes(Nos, N, MatrizDistancia);
    gettimeofday(&time_final,NULL);

    conheceAdjacente(N, MatrizDistancia, Nos);
    conhece_metrica(N, Nos);

    gettimeofday(&time_inicial, NULL);
    tIni = (double) time_inicial.tv_usec / 1000000 + (double) time_inicial.tv_sec;
    for(j = 0; j < N; j++)
    {
        dijkstra(&Nos[j], Nos, N);
    }
    gettimeofday(&time_final, NULL);
    tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;

    gettimeofday(&time_final, NULL);
    tFim = (double) time_final.tv_usec / 1000000 + (double) time_final.tv_sec;

    med_temp_dijks += (tFim - tIni); /*variavel global, para acumular o tempo da
execucao do algoritmo de dijkstra e calculo de rota */
    qt_temp_dijks++;/*variavel global para controlar quantas vezes foi rodado o algoritmo
e encontrado o caminho, consideramos como quantidade de ciclos tambem */

    /*file_rec_matrix("MatrizDistancia", MatrizDistancia);
    file_rec_matrix_dij("Dijkstra");
    file_rec_matrix_met("Metricas");*/

}

/*funcao que de fato cria os dispositvos(circulos)*/
void desenharCirculo(GLint x, GLint y, GLint raio, int num_linhas, double R, double G,
double B) {
        double angle;
        int i,j;

        glColor3f(R,G,B);
        glBegin(GL_LINE_LOOP);
                for (i = 0; i < num_linhas; i++) {
                        angle = 2*PI*i/num_linhas;
```

```
                    glVertex2f((cos(angle)*raio) + x, (sin(angle)*raio) + y);
                }
        glEnd();
}

void organiza()
{
   int i;

   for(i = 0; i < N; i++)
   {
      switch(Nos[i].color){
         case 0:/*dispositivo qualquer, cor preta */
            desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 0.0, 0.0, 0.0);
         break;

         case 1:/*dispositivo inicial, cor verde */
            desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 0.0, 1.0, 0.0);
         break;

         case 2:/*dispositivo final, cor azul */
            desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 0.0, 0.0, 1.0);
         break;

         case 3: /*dispositivo da rota dos dados, rosado */
            desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 1.0, 1.0, 1.0);
         break;

         case 4:/*dispositivo descarregado, vermelho */
            desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 1.0, 0.0, 0.0);
         break;
      }
   }
}

/*funcao para analisar e executar uma acao correspondente as opcoes do menu */
void GoMenu(int value){

   switch(value){
      case 1:
         Nos[dispMenu].color = 1;
         if( click == -1){
            inicial = dispMenu;
            click++;
         }else{
```

```
            Nos[inicial].color = 0;
         }
      break;

      case 2:
         if(click == 0){
            Nos[dispMenu].color = 2;
            click++;
         }
      break;

      case 3:
         Nos[dispMenu].Pwa = Nos[dispMenu].Pwi;

   }
   dispMenu = -1;
   glutPostRedisplay();
}

/*funcao para criacao do menu se o usuario clicar em um dispositivo */
void menu(){
   int sub1;

   printf("click\n");
   glutCreateMenu(GoMenu);
   glutAddMenuEntry("Define source", 1);
   glutAddMenuEntry("Define sink", 2);
   glutAddMenuEntry("Recharge device", 3);
   glutAttachMenu(GLUT_LEFT_BUTTON);
}

// Função callback chamada para gerenciar eventos do mouse
void GerenciaMouse(int button, int state, int x, int y)
{
        int i = 0, x1, y1;
        int encontrou = 0;
        float distancia;

        x1 = x;
        y1 = ay - y;
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        while(i < N && !encontrou){
                            distancia = calcDist(x1, y1, Nos[i].x, Nos[i].y);
                            if(distancia <= 5.0){
              printf("click");
```

```
                    encontrou = 1;
          dispMenu = i;

          menu();
                         //desenharCirculo((GLint)Nos[i].x, (GLint)Nos[i].y, 5,10, 0.0,
1.0, 0.0);
                  }
                  i++;
      }
   }
}

// Função callback chamada para fazer o desenho
void Desenha(void)
{
   glClearColor(0.3, 0.3, 0.3, 0); // sets the backgraound color to black
   glClear(GL_COLOR_BUFFER_BIT); // clears the frame buffer and set values defined
in glClearColor() function call
   glLoadIdentity();

        gluOrtho2D(0.0, ax, 0.0, ay);
        //Limpa a janela de visualização com a cor de fundo especificada

        organiza();

   //glutSwapBuffers();
   glFlush();
}




int main(int argc, char **argv)
{

   pthread_t threads;
   int rc,i;
   /*variáveis para thread de teste*/

   if(argc != 4)
   {
      printf(" Error\n Forma de execucao: %s <num_sensores> <inicial_int> <final_int>\
n", argv[0]);
      exit(1);
   }
```

```
    N = atoi(argv[1]);
    inicial = atoi(argv[2]);
    final = atoi(argv[3]);

    printf("%d %d %d", N, inicial, final);

    glutInit(&argc, argv);
        glutInitDisplayMode( GLUT_RGB | GLUT_SINGLE );
        glutInitWindowSize(ax,ay);
        glutCreateWindow("Dijkstra Based Router Protocol");
        glutDisplayFunc(Desenha);
    glutIdleFunc(Desenha);
    init();
    //glutMouseFunc(GerenciaMouse);
    rc = pthread_create(&threads, NULL, print_time, NULL);

    if(rc){
      printf("Error, return code from pthread_create() is %d\n",rc);
      exit(-1);
    }

        glutMainLoop();

    free(Nos);

    return 0;
}
```

ANEXO A – Algoritmo do Dijkstra padrão

Um dos algoritmos utilizados para obter o caminho mais curto é o algoritmo de Dijkstra, alcançando-o partir de uma origem dentro de um grafo ponderado, em que os pesos contidos nas arestas são maiores ou iguais a zero. Ele mantém um conjunto de vértices chamado de S, em que os pesos finais dos menores caminhos contendo esses vértices, desde a origem já foram determinados. O algoritmo sempre seleciona o vértice de menor peso dentro do conjunto Q, como segue na implementação a seguir:

DIJKSTRA(G, w, s)

1 INITIALIZE-SINGLE-SOURCE(G,s)

2 S ← ∅ // S é o conjunto de vértices já verificados

3 Q ←V[G] //Q é o conjunto total de vértices do grafo

4 **while** Q ≠ ∅

5 **do** u ←EXTRACT-MIN(Q)

6 S ←S ⋃ {u}

7 **for** cada vértice v ∈ Adj[u]

8 **do** RELAX(u, v, w)

Das linhas 1-3 ocorre todas as inicializações do algoritmo, como dos valores d e **pi,** que são respectivamente a distância mínima para alcançar aquele vértice de uma origem chamada de s, em que essa origem é inicializada com zero, e o vértice que antecede o atual, onde a origem não é sucessor de nenhum outro, inicializa também o conjunto S com o conjunto vazio, e por fim inicializa uma fila Q, sendo essa de prioridade mínima,

para abranger todos os vértices em V. Nas linhas de 4 a 8, o invariante de que Q = V - S se mantém em cada iteração do loop(repetição) while, e conforme o código, a cada passo dentro do próprio loop, um vértice u é extraído de Q e introduzindo em S, mantendo o invariante. O vértice retirado em Q, tem menor estimativa em comparação com os demais vértices em V - S. Após isso, nas linhas 7 e 8, relaxam cada vizinho v de u, modificando assim o d[v] e o predecessor **pi**[v] sendo u.

“[…] se o caminho mais curto até v pode ser melhorado mediante a passagem por u. Observe que os vértices nunca são inseridos em Q após a linha 3, e que cada vértice é extraído de Q e inserido em S exatamente uma vez, de modo que o loop **while** das linhas 4 e 8 interage exatamente |V| vezes. ”