Redes Sem Fio

Grupo:

- Renan Almeida Goes Vieira de Melo, 20170021539
- Lucas Moreira e Silva Alves, 20170027336
- Júlio Raphael de Oliveira Silva, 2016085924

Código no GitHub:

https://github.com/JRaphaelO/WirelessNetworks

Introdução

O projeto tem como ideia a implementação de uma simulação de uma rede sem fio, onde pacotes são enviados a partir de uma origem até um destino, passando por nós intermediários.

Para isso, foi decidido implementar o algoritmo de roteamento *Dynamic Source Routing*, pois como já foi estudado anteriormente e sua implementação aparentava mais simples do que os demais protocolos, o desenvolvimento se deu mais célere e eficaz.

Conforme pedido foi feita a simulação da camada física, em que é feita a transmissão dos pacotes em broadcast, de enlace, em que é feito o controle de acesso ao meio e de redes, em que o protocolo utilizado é executado.

Metodologia

Para a implementação do projeto foi utilizada a linguagem de programação Python, aplicando conceitos de programação orientada à objetos ao longo das diferentes camadas de rede.

Visando gerar os gráficos das posições iniciais dos nós da rede, utilizamos a biblioteca *Matplotlib*.

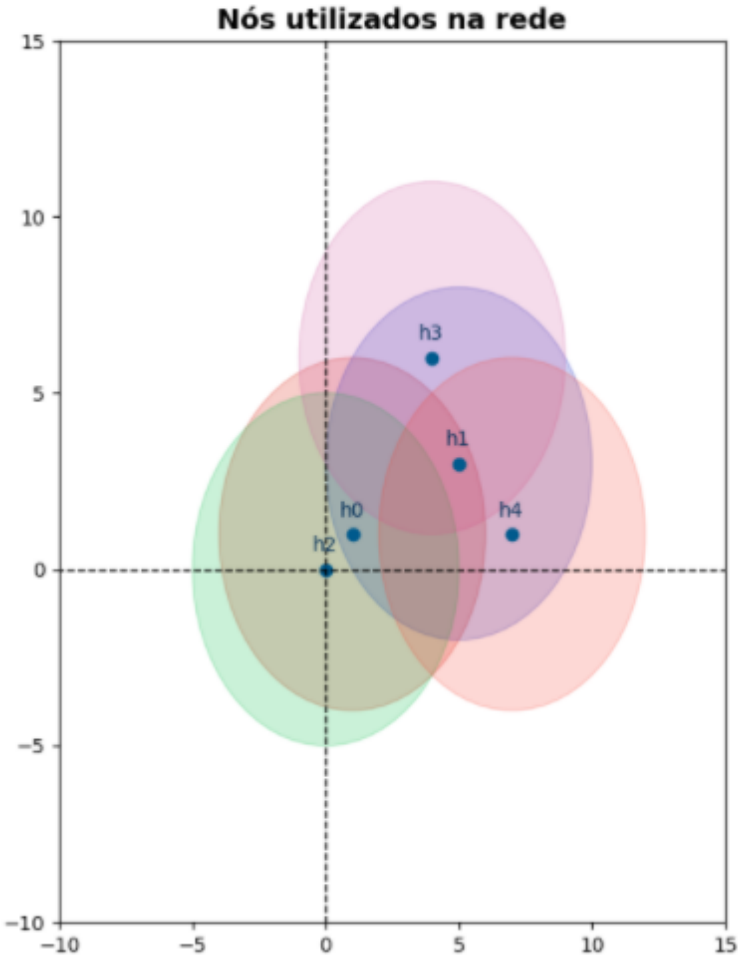


Figura 1: Exemplo de gráfico gerado, dadas as configurações iniciais de posição e nós da rede

Ademais, para calcular a distância euclidiana entre os nós foi usada a biblioteca NumPy, utilizando a estrutura de dados *NumPy.Array* e para o cálculo da distância a função *linalg*.

Como editor de texto usado no projeto utilizamos o Visual Studio Code, e, através da extensão LiveShare pudemos realizar o projeto em conjunto remotamente, editando o mesmo projeto em diferentes computadores. Para o controle de versões utilizamos GitHub.

Como entrada para teste da aplicação são utilizados dois arquivos, um chamado `nodes.txt`, que vai conter, respectivamente da primeira a última linha:

- xmin e ymin do gráfico representando a posição inicial dos nós

- xmax e ymax do gráfico representando a posição inicial dos nós
- Quantidade de nós
- Novas linhas para cada um dos nós da rede, contendo, em cada um deles, posição X, posição Y e o seu raio

Ademais, também é usado como input um arquivo `messages.csv`, que terá 3 colunas: `message`, `host_send` e `host_receive`. Representando, a mensagem que será transmitida, o nó que está enviando e o nó que está recebendo.

Protocolo de roteamento

Para a implementação do *DSR* foi definido que ao um nó enviar um pacote, ele deve enviar para todos os seus vizinhos, que são observados a partir da distância euclidiana entre os nós, se o destino estiver entre seus vizinhos, ele envia a mensagem passando pela camada de enlace e física, senão ele busca se o caminho já conhecido e envia a mensagem, caso o caminho ainda não seja conhecido, cada nó vizinho manda em broadcast uma requisição de rota para os vizinhos deles, que a observam e, para evitar colisões, percebem se uma requisição desse pacote já não havia chegado antes naquele roteador e caso não não houvesse chegado e ele não seja o destino, repete o mesmo processo e envia requisições e broadcast para seus vizinhos, até que chegue no destino ou todos os roteadores até qual a origem possua uma rota tenham recebido a requisição. Caso a requisição chegue no destino, ele envia uma resposta até o nó de origem, que ao receber a resposta envia o pacote com a mensagem até o destino com a rota adquirida.

1. Pacote é enviado em broadcast da origem para seus vizinhos
2. Se um dos vizinhos for o destino o pacote é enviado
3. Se nenhum dos vizinhos forem o destino e o caminho até o destino não é conhecido, uma requisição de rota é enviada em broadcast pelos vizinhos
4. Se o destino for encontrado, um resposta com a rota é enviada até a origem
5. Se o destino não for encontrado, o processo de envio de requisição em broadcast é repetido até todos os nós terem obtido a requisição desse pacote ou o destino seja encontrado
6. Quando a resposta chegar a origem, ela envia o pacote com a mensagem até o destino com a rota adquirida

Componentes da rede

- **Host** → Representa todos os nós que recebem alguma informação e todos que enviam alguma informação, contendo instâncias das camadas de enlace, física e de rede. Possui métodos relacionados ao estabelecimento de vizinhos, checar se o destino está em uma das rotas já realizadas e envio de mensagens
- **Physical**→ A camada física será responsável pelo envio dos pacotes para todos os vizinhos que estão em seu alcance, bem como um método para o recebimento físico do pacote
- **Link**→ É responsável pelas requisições para os caminhos da rede. Envio de pacote para a camada física, recebimento de pacote da camada física para enviar para a camada de rede
- **Message →** Pequeno componente da rede contendo uma mensagem, uma origem e um destino como atributos
- **Node** → Cada nó da rede vai possuir um ID, uma posição x e y no plano cartesiano e um raio
- **Package** → Classe responsável pelos atributos dos pacotes, cada pacote possuirá um id, tipo, conteúdo, origem, destino, próximo nó (inicializado a priori como *None*) e o caminho percorrido por ele (inicializado a priori como uma lista vazia)
- **Package_ID_Counter** → Contagem dos pacotes enviados
- **RequestController** → Vai realizar o controle das requisições, adicionando elementos para a fila de requisições e realizando as tratativas para os nós que podem ou não fazer o envio, bem como retornar todas as requisições
- **Network** → Contem a implementação da rede seguindo o algoritmo de DSR. Realizando a comunicação entre as camadas e orquestrando o controle de envio de pacotes, requisições e respostas na rede

Hospedeiro

Foi implementada uma classe *Host* que é utilizada para controlar o envio e recebimento de pacotes, possuindo dentro dele as camadas física, de enlace e

de rede, além de um controlador de requisições, um conjunto com seus vizinhos conhecidos, uma lista de todos os roteadores da rede e um dicionário de todas as rotas conhecidas por este hospedeiro, com o destino como chave e o valor como a rota em si.

Assim, os métodos desta classe são:

- *set_hosts* → Gera a lista de roteadores da rede
- *send_message* → Chama a função da camada de rede de mandar um pacote (*send_package*) com a mensagem como dado
- *check_destiny* → Retorna verdadeiro ou falso se existe uma rota para um determinado destino que já é conhecida pelo hospedeiro
- *set_neighbours* → Gera o conjunto (*set*) de vizinhos conhecidos a partir da distância euclidiana entre os nós (utilizando *numpy*) e os roteadores da rede

Camada Física

Na camada recebemos como atributo o hospedeiro, que é obtido também pela camada de enlace, herdando então todos os atributos da classe.

Sua função é de enviar pacotes em broadcast para outras interfaces físicas, para isso foram realizadas dois métodos:

- *send_package* → São checados quais roteados são os vizinhos deste a partir da distância euclidiana, o pacote é então enviado para todos os vizinhos observados, ou seja, em broadcast
- *receive_package* → Chama função da camada de enlace para recebe o pacote que tem como parâmetro

Camada de Enlace

Na camada de enlace temos como atributos o próprio hospedeiro, o pacotes pendentes e a quantidade de pacotes. Sua função é o controle de acesso a camada física, portanto os métodos implementadas foram:

- *sending_request* → A requisição é enviada para o objeto de controle de requisições da classe *Host*, adicionando-a a uma fila de requisições,

observando então se a mesma requisição não chegará novamente para evitar colisões

- *send_package_to_physical* → Chama a função *send_package* da camada física para envio do pacote
- *receive_package_to_physical* → Chama a função da classe *Host* de recebimento de pacote

Camada de Rede

A camada de rede possui como atributos o *id* do pacote que está sendo enviado/recebido, uma lista de pacotes recebidos, uma lista de pacotes pendentes e o próprio hospedeiro.

Assim, a camada de rede fica responsável pela implementação do algoritmo *DSR* em si, por isso os métodos utilizados são:

- *get_next_jump* → Retorna o roteador seguinte na rota
- *add_received_package* → Adiciona o pacote a lista de pacotes recebidos caso ainda não esteja nela
- *send_package* → Envia o pacote à camada de enlace caso o destino esteja entre seus vizinhos, caso contrário envia pela camada de enlace requisições em broadcast (*sending_request*), adicionando então à lista de pacotes pendentes
- *receive_rreq_package* → Recebe uma requisição e restorna uma resposta pela camada de enlace caso um dos vizinhos seja o destino ou envia outras requisições pela camada de enlace para os vizinhos
- *receive_rrep_package* → Caso um dos vizinhos seja o destino, elimina o pacote da lista de pacotes pendentes, adiciona a rota as rotas conhecidas pelo hospedeiro e envia o pacote de dados pela camada de enlace, caso contrário envia a resposta pela camada de enlace para o próximo nó da rota

Controlador de requisições

Por fim, foi criada uma classe para o controle das requisições RREQ, que possui como atributos uma lista com os roteadores que desejam mandar uma

requisição e qual o próximo hospedeiro a enviar uma requisição. Com isso, os métodos da classe são:

- *send_permission* → Obtem o hospedeiro a enviar uma requisição fazendo um *pop* na lista de roteadores em espera e envia o pacote da requisição a partir da camada de enlace
- *add_queue* → Adiciona um novo hospedeiro em espera à lista de roteadores em espera para envio de requisição
- *get_all_request* → Retorna a lista de hospedeiros em espera para envio de requisição

Resultados da Simulação

Para a realização da simulação, foram realizados 3 casos, que nos quais são: o hospedeiro conhece o nó de destino, o hospedeiro não conhece o nó de destino e por fim o hospedeiro conhece uma rota até o nó de destino.

Para isso, foram utilizadas as figuras 2, 3 e 4, para demonstrar os comportamentos da nossa simulação, sendo assim temos que a cor azul representa quem envia a mensagem, a cor laranja representa o host de destino, a cor cinza representa os outros hosts.

1º Caso: O hospedeiro conhece o nó de destino:

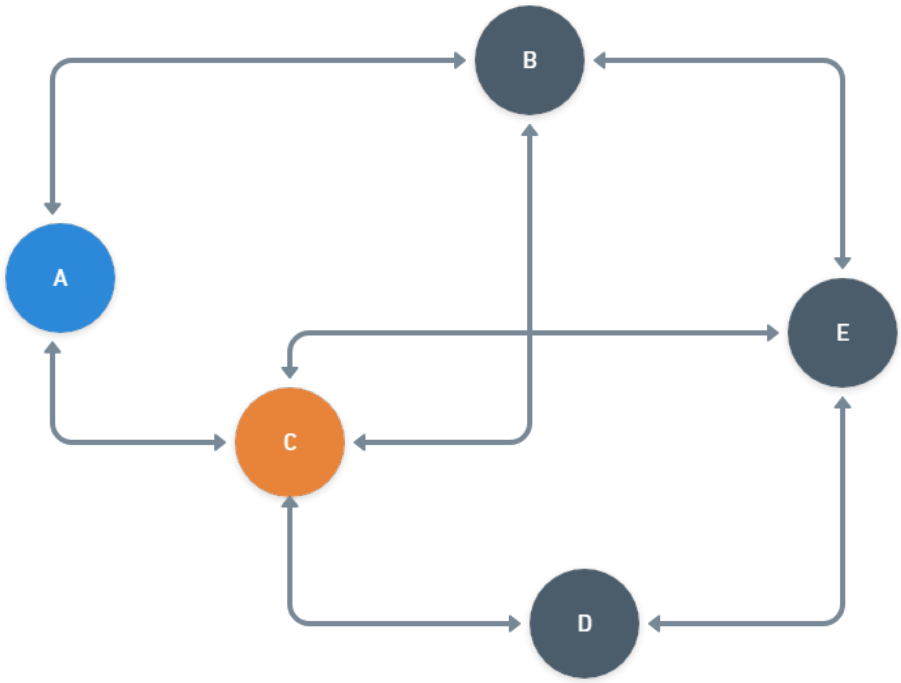


Figura 2: O host A envia uma mensagem para o C

Para esta simulação o hospedeiro A, enviará a mensagem "Olá host C", para o hospedeiro C, com isso temos que os logs desta simulação, se dá por:

```
INFO:root:Intial Time: 05/07/2021 16:04
INFO:root:     Network: create DATA package from host[0] to destination host[1]
INFO:root:     Network: host[1] is my neighbour
INFO:root:     Link: sending package to request controller from host[0]
INFO:root:     Request Controller: the Host[0] want to send a package, wait is permission to send.

INFO:root:  1. Time Request: 05/07/2021 16:04
INFO:root:     Request Controller: Host[0] is allowed to start sending
INFO:root:     Physical: Host[0] sending package to Host[2]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Physical: Host[0] sending package to Host[1]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Network: host[1] receive a package from host[0]
                        Message is: Olá host C.
INFO:root:
```

Hospedeiro observa que um de seus vizinhos é o destino e envia o pacote de dados a ele

2º Caso: O hospedeiro não conhece uma rota até o nó de destino

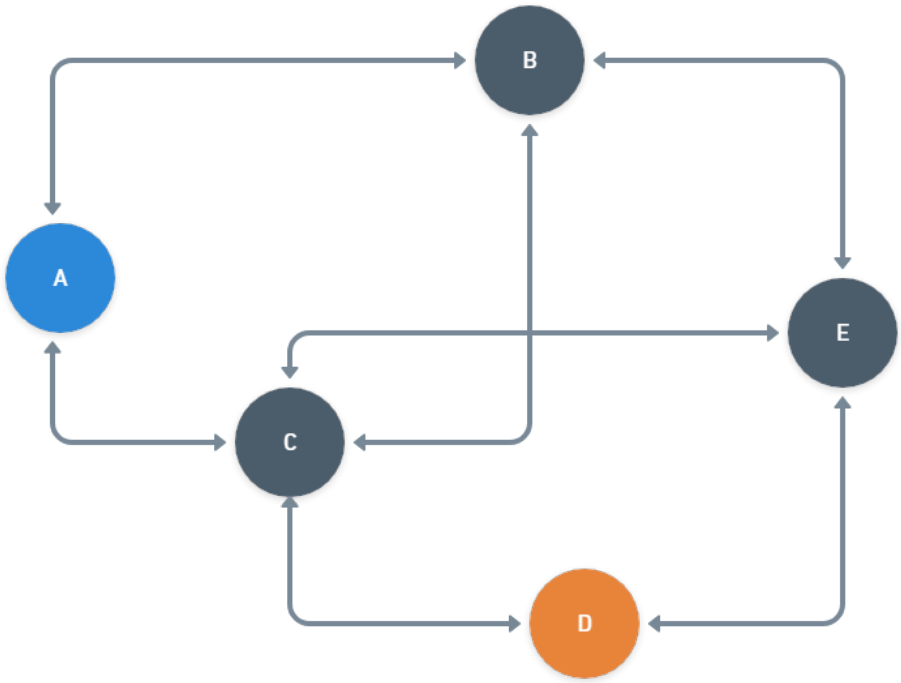


Figura 3: O host A envia uma mensagem para o D sem conhecer o caminho.

Para esta simulação o hospedeiro A, enviará a mensagem "Olá host D", para o hospedeiro D, com isso temos que os logs desta simulação, se dá por:

```
INFO:root:     Network: create DATA package from host[0] to destination host[3]
INFO:root:     Network: host[0] has no routes to host[3]
INFO:root:     Network: host[0] is creating a RREQ package to the host[3]
INFO:root:     Network: host[0] is adding itself to the path
INFO:root:     Link: sending package to request controller from host[0]
INFO:root:     Request Controller: the Host[0] want to send a package, wait is permission to send.
```

O pacote de dados é criado, ao ver que não há uma rota ao destino, nem o destino é algum dos vizinhos, o hospedeiro manda uma requisição RREQ em broadcast.

```
INFO:root:  3. Time Request: 05/07/2021 16:11
INFO:root:     Request Controller: Host[2] is allowed to start sending
INFO:root:     Physical: Host[2] sending package to Host[3]
INFO:root:     Link: receive package |RREQ| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |RREQ| by Link Layer.
INFO:root:     Network: host[3] has received a RREQ package, which is the destination
INFO:root:     Network: host[3] is sending a RREP package to host[0]
INFO:root:     Link: sending package to request controller from host[3]
INFO:root:     Request Controller: the Host[3] want to send a package, wait is permission to send.
```

A requisição chega no destino, o qual manda uma resposta pelo caminho reverso obtido para chegar nele.

```
INFO:root:  7. Time Request: 05/07/2021 16:11
INFO:root:     Request Controller: Host[2] is allowed to start sending
INFO:root:     Physical: Host[2] sending package to Host[3]
INFO:root:     Link: receive package |RREP| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |RREP| by Link Layer.
INFO:root:     Physical: Host[2] sending package to Host[0]
INFO:root:     Link: receive package |RREP| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |RREP| by Link Layer.
INFO:root:     Network: host[0] has received a RREP package from the host[3]
INFO:root:     Network: I am the origin, and I'm sending a DATA package to host[3]
INFO:root:     Network: I'm Sending DATA to next Host[2]
INFO:root:     Link: sending package to request controller from host[0]
INFO:root:     Request Controller: the Host[0] want to send a package, wait is permission to send.
```

A resposta chega na origem, que manda um pacote de dados pela rota obtida no pacote RREP de resposta

```
INFO:root:  9. Time Request: 05/07/2021 16:11
INFO:root:     Request Controller: Host[2] is allowed to start sending
INFO:root:     Physical: Host[2] sending package to Host[3]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Network: host[3] receive a package from host[0]
                        Message is: Olá host D.
INFO:root:     Physical: Host[2] sending package to Host[0]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Physical: Host[2] sending package to Host[4]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Physical: Host[2] sending package to Host[1]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:
```

O pacote de dados com a mensagem chega no destino a partir da rota obtida

3º Caso: O hospedeiro conhece uma rota até o nó de destino

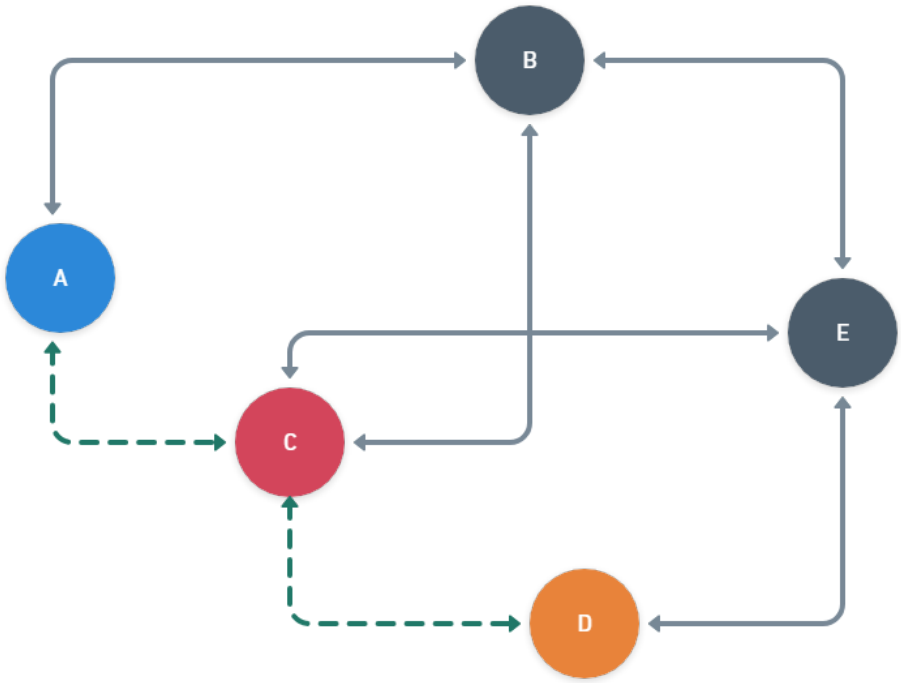


Figura 4: O host A envia uma mensagem para D e conhecendo o caminho.

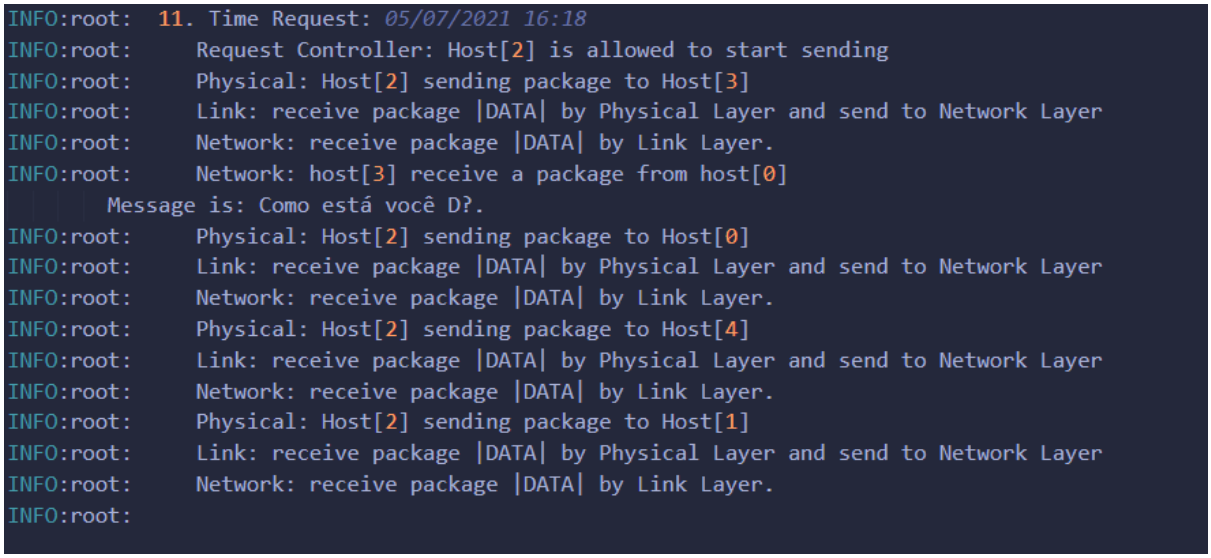
Para esta simulação o hospedeiro A, enviará a mensagem "Como está você D?", para o hospedeiro D, com isso temos que os logs desta simulação, se dá por:

```
INFO:root:     Network: create DATA package from host[0] to destination host[3]
INFO:root:     Network: host[0] can get into the destination
INFO:root:     Network: I'm Sending DATA to next Host[2]
INFO:root:     Link: sending package to request controller from host[0]
INFO:root:     Request Controller: the Host[0] want to send a package, wait is permission to send.
```

O hospedeiro observa que existe uma rota para o destino e manda o pacote de dados

```
INFO:root:  10. Time Request: 05/07/2021 16:18
INFO:root:     Request Controller: Host[0] is allowed to start sending
INFO:root:     Physical: Host[0] sending package to Host[2]
INFO:root:     Link: receive package |DATA| by Physical Layer and send to Network Layer
INFO:root:     Network: receive package |DATA| by Link Layer.
INFO:root:     Network: host[2] receives a DATA package, but I'm not the destination
INFO:root:     Network: I'm Sending DATA to next Host[3]
INFO:root:     Link: sending package to request controller from host[2]
INFO:root:     Request Controller: the Host[2] want to send a package, wait is permission to send.
```

O pacote de dados passa pelos nós intermediários para chegar ao destino



O pacote de dados chega no destino

Conclusão

Ao longo do projeto, foi possível compreender muito mais a fundo o funcionamento das diferentes camadas implementadas em um projeto de redes sem fio, bem como os tratamentos necessários para o correto funcionamento do fluxo da rede.