Trabalho 1 de Redes Avançadas

Alunos: Gabriel Tabajara, Giovanni Schenato, Guilherme Romanini e Lucas Dellatorre.

1. Detalhes de implementação (linguagem, classes, principais métodos)

O código do trabalho foi escrito na linguagem python e possui 8 arquivos:

- main.py
  - Arquivo inicial da aplicação, invoca o FileReader e o ExecManager.
- file_reader.py
  - Responsável por ler a topologia do arquivo ".txt" e criar o backbone com as instâncias das classes (Router, Subnet)
  - Adiciona as linhas (registros) nas tabelas dos roteadores
  - Lê o arquivo de execução e transforma em ExecInstructions para o ExecManager entender.
- backbone.py
  - Classe responsável por armazenar os objetos que compõem a topologia da rede, roteadores e subnets.
  - Possui um método para achar uma subnet com base no seu id.
- exec_manager.py
  - Arquivo responsável por executar as instruções de forma sequencial.
    - Extrai a subnet do backbone e invoca seu método referente ao nome da instrução, *mping*, *mjoin* ou *mleave*.
- router.py
  - Implementação da classe Router.
  - Principais atributos:
    - id, interfaces, interfaces_num.
    - routerTable: Array com todas as entradas da tabela como RouterTableRegistry .
    - groupTable: Dicionário que tem como chave o id do grupo e como valor um array com as subnets que estão dentro do respectivo grupo.
  - Principais métodos:
    - mjoin: adiciona uma subnet a um grupo.
    - mleave: remove uma subnet de um grupo.
    - mpingStarter: Recebe o mping de uma subnet, cria o pacote com informações importantes, chama o método

de flood, caso esteja ligado a algum roteador e após acabar o flood e o prune invoca o *mping*.
        - mflood: implementa o *mflood* fazendo com que os roteadores mandem os pacotes de inundação uns para os outros, respeitando o RPF. O *mflood* possui 2 métodos auxiliares, o mfloodStart e o mfloodReceive. Esse retorna uma lista com os roteadores que recebem a mensagem (esses podem ser intermediários ou roteadores que possuem subnets no grupo destinatário)
        - mfloodReceive: Análise se o roteador mantém ou descarta o pacote baseado na técnica RPF.
        - mfloodStart: É chamado quando "aceita" o flood, sua função é continuar o flood a partir dele (do roteador que teve esse método chamado) verifica se deve enviar um *mprune*.
        - mprune: Codifica a mensagem que bloqueia o tráfego multicast.
        - mping: Responsável por exibir o caminho do *mping*, do roteador de origem, até as subnets que estão no grupo destino, chama também o *mrecv*.
        - mrecv: Exibe a mensagem *mrecv*.

- subnet.py
    - Implementação da classe Subnet, no construtor é atribuido seu ID e endereço.
    - Possui um mainRouter, cuja função é encaminhar uma mensagem caso seja o emissor.
    - Responsável por mandar as mensagens ao roteador ao qual se conecta (mainRouter)
        - métodos: *mping*, *mjoin* e *mleave*
- router_table.py
    - Implementação da classe RouterTableRegistry, um registro de uma tabela de roteador, possui apenas um construtor e seus atributos
        - netaddr: endereço da rede destino.
        - next_hop: endereço do próximo salto.
        - interface_num: número da interface do roteador.
- exec_instructions.py
    - Implementação da classe ExecInstructions, possui apenas um construtor e seus atributos
        - cmd
        - subnet
        - group
        - msg (pode ser nula)

2. Descrição de como utilizar o simulador com exemplo de execução não visto em aula

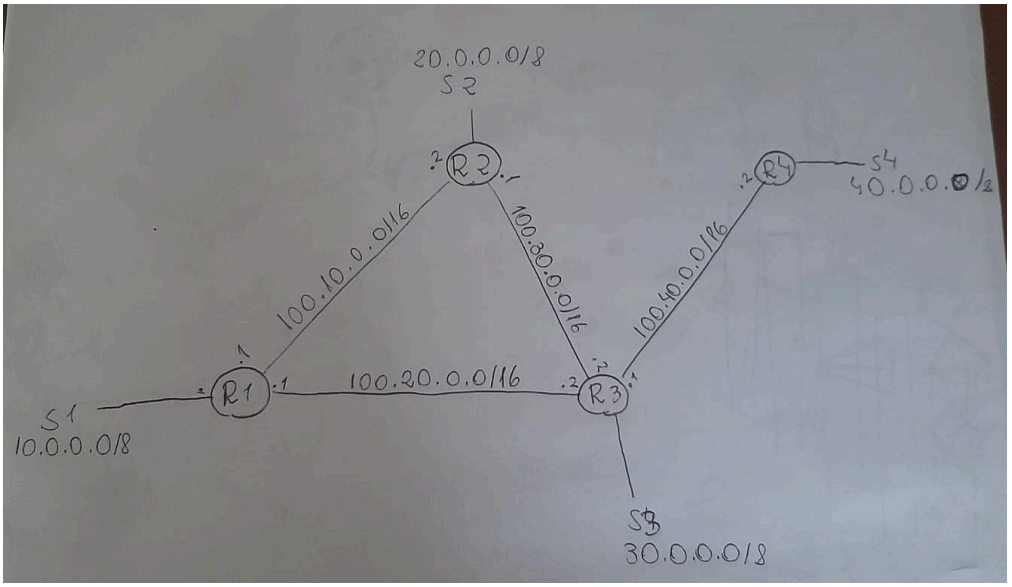
Para utilizarmos o nosso simulador utilizamos o seguinte comando para rodar o projeto:

```
python main.py topologia.txt exec.txt
```

Sendo topologia.txt o nome do arquivo da topologia e exec.txt o nome do arquivo com os comandos de execução.

Abaixo, teremos um exemplo de execução com uma topologia não vista em aula:

Topologia:



```
#SUBNET
s1,10.0.0.0/8
s2,20.0.0.0/8
s3,30.0.0.0/8
s4,40.0.0.0/8
#ROUTER
r1,3,10.0.0.1/8,100.10.0.1/16,100.20.0.1/16
r2,3,100.10.0.2/16,100.30.0.1/16,20.0.0.1/8
r3,4,100.30.0.2/16,100.40.0.1/16,100.20.0.2/16,30.0.0.1/8
r4,2,100.40.0.2/16,40.0.0.1/8
#ROUTERTABLE
r1,10.0.0.0/8,0.0.0.0,0
r1,20.0.0.0/8,100.10.0.2,1
r1,30.0.0.0/8,100.20.0.2,2
r1,40.0.0.0/8,100.20.0.2,2
r2,10.0.0.0/8,100.10.0.1,1
r2,20.0.0.0/8,0.0.0.0,0
r2,30.0.0.0/8,100.30.0.2,2
r2,40.0.0.0/8,100.30.0.2,2
r3,10.0.0.0/8,100.20.0.1,1
r3,20.0.0.0/8,100.30.0.1,2
r3,30.0.0.0/8,0.0.0.0,0
r3,40.0.0.0/8,100.40.0.2,1
r4,10.0.0.0/8,100.40.0.1,1
r4,20.0.0.0/8,100.40.0.1,1
r4,30.0.0.0/8,100.40.0.1,1
r4,40.0.0.0/8,0.0.0.0,0
```

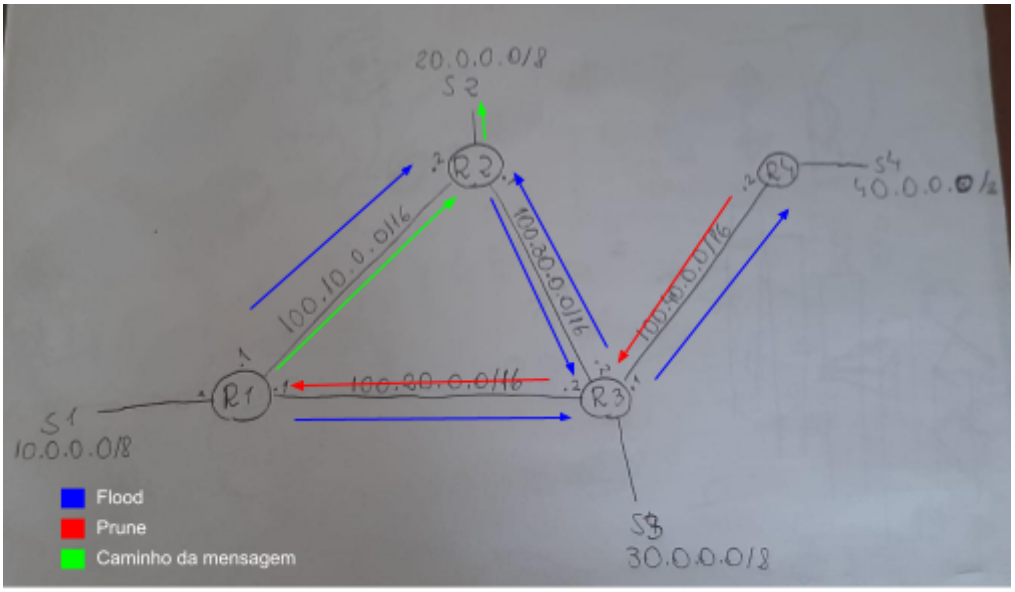
Exec:

```
mjoin s2 g1
mping s1 g1 Hello
mjoin s4 g1
mping s1 g1 World
mleave s2 g1
mleave s4 g1
```

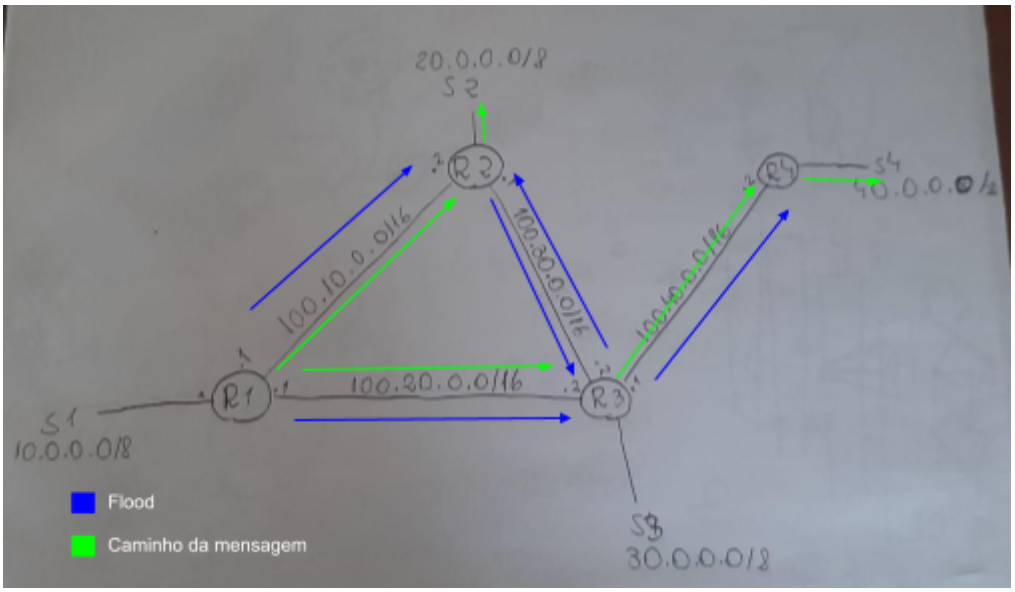
Resultado após execução:

```
s2 => r2 : mjoin g1;
s1 =>> r1 : mping g1 Hello;
r1 >> r2, r1 >> r3 : mflood g1;
r2 >> r3 : mflood g1;
r3 >> r2, r3 >> r4 : mflood g1;
r4 >> r3 : mprune g1;
r3 >> r1 : mprune g1;
r1 =>> r2 : mping g1 Hello;
r2 =>> s2 : mping g1 Hello;
s2 box s2 : g1#Hello from s1;
s4 => r4 : mjoin g1;
s1 =>> r1 : mping g1 World;
r1 >> r2, r1 >> r3 : mflood g1;
r2 >> r3 : mflood g1;
r3 >> r2, r3 >> r4 : mflood g1;
r1 =>> r2, r1 =>> r3 : mping g1 World;
r2 =>> s2 : mping g1 World;
s2 box s2 : g1#World from s1;
r3 =>> r4 : mping g1 World;
r4 =>> s4 : mping g1 World;
s4 box s4 : g1#World from s1;
s2 => r2 : mleave g1;
s4 => r4 : mleave g1;
```

Topologia após o “mping s1 g1 Hello”:



Topologia após o “mping s1 g1 World”:



3. Limitações do simulador implementado

Abaixo segue uma lista com as limitações do simulador implementado:

- A subnet não pode se conectar a dois roteadores.
- Não garante que o mesmo roteador receba apenas um pacote mflood, mas garante que um roteador encaminhe apenas uma cópia do mflood que recebeu, as outras são descartadas.

4. Dificuldades de implementação.

Uma das dificuldades foi realizar o algoritmo de reverse path forwarding, visto que havia exemplos que nossa implementação não previa, então tivemos que corrigir para suportar demais exemplos.