INTRODUÇÃO A REDES

Trabalho Final - Simulador de Rede

Diogo Hartmann e Júlia Melgaré

1. Detalhes de Implementação

A linguagem utilizada foi Python, implementamos 4 classes, classe Node implementando o Nodo ou Host contendo seu IP, MAC, MTU, Gateway; a classe Router implementando o Roteador contendo IPs, MACs e MTUs para cada porta do roteador e a Tabela de Roteamento sendo um dicionário de IP para Entrada de Roteador ; a classe RouterTableEntry implementando uma Entrada de Roteador contendo o próximo pulo e a interface de saída; classe Message que implementa a Mensagem em si contendo MAC origem, MAC destino, IP origem, IP destino, tipo de Mensagem que pode ser Echo Request, Echo Reply ou Time Exceeded, Time to Live(TTL), o campo More Fragments indicando se há mais fragmentos da mensagem após esta, o campo offset para reconstruir a mensagem e por fim o campo de dados. O Node e Router contém também uma tabela cache de endereços para ARP.

Métodos do Nodo ou Host

SendPackage: Primeiramente temos a requisição de um ARP request caso o endereço destino não esteja contido no cache de ARPs, se o endereço IP destino pertence a mesma rede o nodo destino responde a requisição com seu MAC, caso contrário o roteador da rede responde com o MAC da porta apropriada. Após sabermos o MAC montamos as mensagens caso haja fragmentação ou somente uma caso contrário e chamamos o método Relay Package para o próximo hop.

RelayPackage: Caso este seja chamado para o Nodo sabemos que este é o Nodo destino final da mensagem, assim somente chamamos SendPackage caso necessite de uma resposta.

Métodos do Roteador

RelayPackage: Este tem a mesma função do Send Package do Nodo, com a diferença que necessitamos checar a Tabela de Roteamento antes para saber para qual porta o pacote é mandado. Novamente requisitamos o endereço MAC por ARP e montamos a(s) mensagem(ns) com TTL - 1 chamando Relay Package. Caso o ttl tenha chegado a 1 reenviamos o pacote ao sender para notificá-lo que TTL Exceeded.

2. Como utilizar

Para utilizar o simulador, basta executar o script em python na linha de comando da seguinte forma:

```
python NetworkSim.py <arquivo de texto da topologia> <nodo de origem> <nodo destino> <mensagem>
```

A implementação do trabalho, bem como as topologias de exemplo utilizadas para desenvolvê-lo estão disponíveis no repositório https://github.com/Diogo45/NetworkSimulator.

3. Limitações e Dificuldades

As partes mais difíceis de implementar no simulador foi a parte de roteamento e de subfragmentação, que é a situação onde temos uma mensagem que já foi fragmentada devido a um MTU pequeno, e que deve ser fragmentada novamente devido a um MTU ainda menor.

4. Exemplos de Execução

Ao executar o simulador com a topologia vista na aula do dia 4 de novembro de 2019, que possui 3 roteadores e 6 nodos, obtivemos o seguinte resultado:

```
python NetworkSim.py ex041119.txt N1 N3 helloworld

N1 box N1 : ETH (src=00:00:00:00:00:01 dst=FF:FF:FF:FF:FF:FF) \n ARP -
Who has - 10.0.0.1? Tell 10.0.0.2;
R1 => N1 : ETH (src=00:00:00:00:00:10 dst=00:00:00:00:00:01) \n ARP -
10.0.0.1 is at 00:00:00:00:00:10;
N1 => R1 : ETH (src=00:00:00:00:00:01 dst=00:00:00:00:00:10) \n IP
(src=10.0.0.2/8 dst=20.0.0.2/8 ttl=8 mf=0 off=0) \n ICMP - Echo Request
(data=helloworld);
R1 box R1 : ETH (src=00:00:00:00:00:11 dst=FF:FF:FF:FF:FF:FF) \n ARP -
Who has - 100.10.20.2/24? Tell 100.10.20.1;
R2 => R1 : ETH (src=00:00:00:00:00:21 dst=00:00:00:00:00:11) \n ARP -
100.10.20.2/24 is at 00:00:00:00:00:21;
R1 => R2 : ETH (src=00:00:00:00:00:11 dst=00:00:00:00:00:21) \n IP
(src=10.0.0.2/8 dst=20.0.0.2/8 ttl=8 mf=1 off=0) \n ICMP -  Echo
Request (data=hello);
R1 => R2 : ETH (src=00:00:00:00:00:11 dst=00:00:00:00:00:21) \n IP
(src=10.0.0.2/8 dst=20.0.0.2/8 ttl=8 mf=0 off=5) \n ICMP - Echo Request
(data=world);
R2 box R2 : ETH (src=00:00:00:00:00:20 dst=FF:FF:FF:FF:FF:FF) \n ARP -
Who has - 20.0.0.2? Tell 20.0.0.1;
N3 => R2 : ETH (src=00:00:00:00:00:03 dst=00:00:00:00:00:20) \n ARP -
20.0.0.2 is at 00:00:00:00:00:03;
R2 => N3 : ETH (src=00:00:00:00:00:20 dst=00:00:00:00:00:03) \n IP
(src=10.0.0.2/8 dst=20.0.0.2/8 ttl= 6 mf=0 off=0) \n ICMP - Echo
Request (data=helloworld);
N3 rbox N3 : Received helloworld;
N3 => R2 : ETH (src=00:00:00:00:00:03 dst=00:00:00:00:00:22) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl=8 mf=0 off=0) \n ICMP - Echo Reply
(data=helloworld);
R2 box R2 : ETH (src=00:00:00:00:00:22 dst=FF:FF:FF:FF:FF:FF) \n ARP -
Who has - 100.10.30.2/24? Tell 100.10.30.1;
R3 => R2 : ETH (src=00:00:00:00:00:31 dst=00:00:00:00:00:22) \n ARP -
100.10.30.2/24 is at 00:00:00:00:00:31;
R2 => R3 : ETH (src=00:00:00:00:00:22 dst=00:00:00:00:00:31) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl=8 mf=1 off=0) \n ICMP -  Echo Reply
(data=hel);
R2 => R3 : ETH (src=00:00:00:00:00:22 dst=00:00:00:00:00:31) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl=8 mf=1 off=3) \n ICMP -  Echo Reply
(data=low);
```

```
R2 => R3 : ETH (src=00:00:00:00:00:22 dst=00:00:00:00:00:31) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl=8 mf=1 off=6) \n ICMP -  Echo Reply
(data=orl);
R2 => R3 : ETH (src=00:00:00:00:00:22 dst=00:00:00:00:00:31) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl=8 mf=0 off=9) \n ICMP - Echo Reply
(data=d);
R3 box R3 : ETH (src=00:00:00:00:00:32 dst=FF:FF:FF:FF:FF:FF) \n ARP -
Who has - 100.10.40.1/24? Tell 100.10.40.2;
R1 => R3 : ETH (src=00:00:00:00:00:12 dst=00:00:00:00:00:32) \n ARP -
100.10.40.1/24 is at 00:00:00:00:00:12;
R3 => R1 : ETH (src=00:00:00:00:00:32 dst=00:00:00:00:00:12) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl= 6 mf=0 off=0) \n ICMP - Echo Reply
(data=helloworld);
R1 => N1 : ETH (src=00:00:00:00:00:10 dst=00:00:00:00:00:01) \n IP
(src=20.0.0.2/8 dst=10.0.0.2/8 ttl= 5 mf=0 off=0) \n ICMP - Echo Reply
(data=helloworld);
N1 rbox N1 : Received helloworld;
```