Trabalho final: Simulador de Redes

Jovani O. Brasil[1]

[1]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

jovani.brasil@acad.pucrs.br

1. Desenvolvimento

Para a implementação do simulador foi utilizada a linguagem de programação *Python* versão 2.7. O primeiro passo no desenvolvimento do simulador foi a implementação do módulo chamado *Util*. Este módulo concentra a leitura e o processamento dos dados, assim como a instanciação e armazenamento dos objetos necessários. Para organizar e também facilitar a comunicação entre os objetos do simulador, os objetos envolvidos na simulação precisaram ser modelados no formato de classes. Com isso, foi necessária a implementação de diversas classes. Entre as principais classes estão a *Node*, *Router* e *Packet*. Dentro de cada uma da classes implementadas existem todos os métodos necessários para o processamento e execução das simulações.

1.1. *Node*

O objetivo da classe *Node* é implementar um dispositivo capaz de enviar e receber mensagens. Para isso, um *Node* possui um método *icmp_send(...)* para envio de mensagens e um método *icmp_receive(...)* para recebimento de mensagens.

1.2. *Router*

A classe *Router* implementa um roteador e é responsável pelo roteamento intermediário dos pacotes entre as redes. Ela possui os mesmos métodos que o *Node* para envio e recebimento de mensagens. O gerenciamento de suas várias portas é implementado nas classes *RouterPorts* e *RouterPort*. Sua tabela *ARP* e tabela de roteamento são gerenciados pelas classes *ArpTable* e *RouterTable*.

1.3. Table

O módulo *Table* consiste das implementações das classes *ArpTable* e *RouterTable*. Na classe *ArpTable* é feito o gerenciamento das tabelas *ARP* dos *Hosts* e dos *Routers*. Como exemplo, um método muito utilizado da classe *ArpTable* é o *get_mac(...)* que procura nas entradas da tabela pelo ip desejado e retorna o *MAC* correspondente se existir. Na classe *RouterTable* temos o gerenciamento da tabela de roteamento utilizada pelos *routers*. Por exemplo, o método *get_destiny(...)* que retorna por qual porta o roteador deve enviar um pacote dado ip.

1.4. Packet

Na classe *Packet* é implementado um pacote que deverá ser transmitido entre objetos através de trocas de mensagens. Nela também são implementados as saídas específicas para cada tipo de pacote no formato especificado.

2. Utilização

Para execução do simulador é necessário ter o Python versão 2.7 instalado na máquina. Não é garantido o funcionamento com a versão 3.1. Arquivos de definição das topologias obrigatoriamente devem estar no diretório *data*. Para executar o simulador no terminal de comandos devemos executar o seguinte comando:

$ python Main.py [topologia.txt] [nodo_inicial] [lista_de_nodos_destino]

Estamos chamando a função *Main.py*, passando o nome do arquivo de topologia, definindo o nodo inicial e uma lista de nodos destino. Exemplos:

$ python Main.py topo4.txt n1 n2

$ python Main.py topo4.txt n1 n2 n1 n3

3. Caso de Teste

3.1. Topologia Proposta

Os casos de teste podem ser encontrados no diretório *data*, entre eles está o *topo4.txt* que descreve a topologia mostrada na Figura 1. Observe que no canto superior direito são indicadas as numerações das portas dos roteadores.

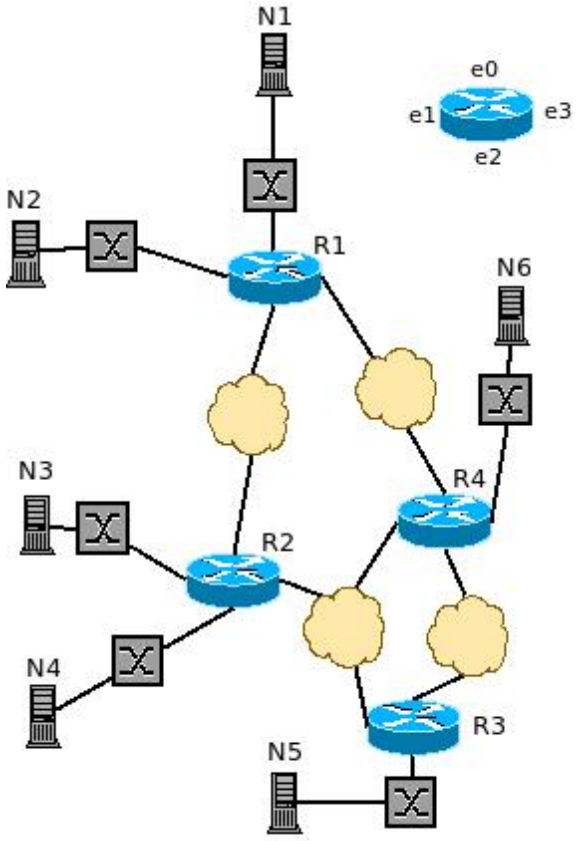

Figura 1. Topologia mais complexa proposta para validação do trabalho.

Seguindo o modelo de entrada proposto as configurações de endereços de portas de roteadores e máquinas são as seguintes:

Tabela 1. Configuração dos hosts.

Host	MAC	IP	Default Gateway
N1	00:00:00:00:00:01	192.168.64.2/18	192.168.64.1
N2	00:00:00:00:00:02	192.168.128.2/18	192.168.128.1
N3	00:00:00:00:00:03	192.168.192.2/18	192.168.192.1
N4	00:00:00:00:00:04	140.64.0.2/12	140.64.0.1
N5	00:00:00:00:00:05	140.96.0.2/12	140.96.0.1
N6	00:00:00:00:00:06	90.20.74.66/27	90.20.74.65

Tabela 2. Configuração dos roteadores.

Router	Port	IP	MAC
R1	0	192.168.64.1/18	00:00:00:00:00:07
R1	1	192.168.128.1/18	00:00:00:00:00:08
R1	2	140.32.0.1/12	00:00:00:00:00:09
R1	3	90.20.74.33/27	00:00:00:00:00:10
R2	1	140.32.0.2/12	00:00:00:00:00:11
R2	2	192.168.192.1/18	00:00:00:00:00:12
R2	3	140.64.0.1/12	00:00:00:00:00:13
R2	4	140.16.0.1/12	00:00:00:00:00:14
R3	0	140.58.0.1/12	00:00:00:00:00:15
R3	1	140.16.0.2/12	00:00:00:00:00:16
R3	2	140.96.0.1/12	00:00:00:00:00:17
R4	0	90.20.74.65/27	00:00:00:00:00:18
R4	1	90.20.74.34/27	00:00:00:00:00:19
R4	2	140.16.0.3/12	00:00:00:00:00:20
R4	3	140.58.0.2/12	00:00:00:00:00:21

Tabela 3. Tabelas de roteamendo.

Tabela 4. Roteador R1.

Destiny	IP	Port
192.168.64.0/18	0.0.0.0	0
192.168.128.0/18	0.0.0.0	1
0.0.0.0/0	140.32.0.2	2

Tabela 5. Roteador R2.

Destiny	IP	Port
192.168.192.0/18	0.0.0.0	1
140.64.0.0/12	0.0.0.0	2
0.0.0.0/0	140.16.0.3	3

Tabela 6. Roteador R3.

Destiny	IP	Port
140.96.0.0/12	0.0.0.0	2
0.0.0.0/0	140.16.0.1	1

Tabela 7. Roteador R4.

Destiny	IP	Port
90.20.74.64/27	0.0.0.0	0
192.168.64.0/18	90.20.74.33	1
192.168.128.0/18	90.20.74.33	1
0.0.0.0/0	140.58.0.1	1

Para inserção do loop na topologia, foram feitas algumas mudanças na tabela de roteamento do roteador R2, como pode ser visto na Tabela 8. Estas mudanças também podem ser vistas na topologia do arquivo *topo5.txt*.

Tabela 8. Roteador R2 com loop.

Destiny	IP	Port
192.168.192.0/18	0.0.0.0	1
140.64.0.0/12	0.0.0.0	2
90.20.74.64/27	140.16.0.2	3
0.0.0.0/0	140.16.0.3	3

3.2. Execução

3.2.1. $ python Main.py topo4.txt n1 n2

```
n1 box n1 : ARP - Who has 192.168.64.1? Tell 192.168.64.2;
r1 => n1 : ARP - 192.168.64.1 is at 00:00:00:00:00:07;
n1 => r1 : ICMP - Echo request (src=192.168.64.2 dst=192.168.128.2 ttl=8);
r1 box r1 : ARP - Who has 192.168.128.2? Tell 192.168.128.1;
n2 => r1 : ARP - 192.168.128.2 is at 00:00:00:00:00:02;
r1 => n2 : ICMP - Echo request (src=192.168.64.2 dst=192.168.128.2 ttl=7);
n2 => r1 : ICMP - Echo reply (src=192.168.128.2 dst=192.168.64.2 ttl=8);
r1 => n1 : ICMP - Echo reply (src=192.168.128.2 dst=192.168.64.2 ttl=7);
```

3.2.2. $ python Main.py topo4.txt n1 n2 n3 n2 n4

```
n1 box n1 : ARP - Who has 192.168.64.1? Tell 192.168.64.2;
r1 => n1 : ARP - 192.168.64.1 is at 00:00:00:00:00:07;
n1 => r1 : ICMP - Echo request (src=192.168.64.2 dst=192.168.128.2 ttl=8);
r1 box r1 : ARP - Who has 192.168.128.2? Tell 192.168.128.1;
n2 => r1 : ARP - 192.168.128.2 is at 00:00:00:00:00:02;
r1 => n2 : ICMP - Echo request (src=192.168.64.2 dst=192.168.128.2 ttl=7);
n2 => r1 : ICMP - Echo request (src=192.168.128.2 dst=192.168.192.2 ttl=8);
r1 box r1 : ARP - Who has 140.32.0.2? Tell 140.32.0.1;
r2 => r1 : ARP - 140.32.0.2 is at 00:00:00:00:00:08;
r1 => r2 : ICMP - Echo request (src=192.168.128.2 dst=192.168.192.2 ttl=7);
r2 box r2 : ARP - Who has 192.168.192.2? Tell 192.168.192.1;
n3 => r2 : ARP - 192.168.192.2 is at 00:00:00:00:00:03;
r2 => n3 : ICMP - Echo request (src=192.168.128.2 dst=192.168.192.2 ttl=6);
n3 => r2 : ICMP - Echo request (src=192.168.192.2 dst=192.168.128.2 ttl=8);
r2 box r2 : ARP - Who has 140.16.0.3? Tell 140.16.0.1;
r4 => r2 : ARP - 140.16.0.3 is at 00:00:00:00:00:12;
r2 => r4 : ICMP - Echo request (src=192.168.192.2 dst=192.168.128.2 ttl=7);
r4 box r4 : ARP - Who has 90.20.74.33? Tell 90.20.74.34;
r1 => r4 : ARP - 90.20.74.33 is at 00:00:00:00:00:20;
r4 => r1 : ICMP - Echo request (src=192.168.192.2 dst=192.168.128.2 ttl=6);
r1 => n2 : ICMP - Echo request (src=192.168.192.2 dst=192.168.128.2 ttl=5);
n2 => r1 : ICMP - Echo request (src=192.168.128.2 dst=140.64.0.2 ttl=8);
r1 => r2 : ICMP - Echo request (src=192.168.128.2 dst=140.64.0.2 ttl=7);
r2 box r2 : ARP - Who has 140.64.0.2? Tell 140.64.0.1;
n4 => r2 : ARP - 140.64.0.2 is at 00:00:00:00:00:04;
r2 => n4 : ICMP - Echo request (src=192.168.128.2 dst=140.64.0.2 ttl=6);
n4 => r2 : ICMP - Echo reply (src=140.64.0.2 dst=192.168.64.2 ttl=8);
r2 => r4 : ICMP - Echo reply (src=140.64.0.2 dst=192.168.64.2 ttl=7);
r4 => r1 : ICMP - Echo reply (src=140.64.0.2 dst=192.168.64.2 ttl=6);
r1 => n1 : ICMP - Echo reply (src=140.64.0.2 dst=192.168.64.2 ttl=5);
```

3.2.3. $ python Main.py topo4.txt n1 n5

```
n1 box n1 : ARP - Who has 192.168.64.1? Tell 192.168.64.2;
r1 => n1 : ARP - 192.168.64.1 is at 00:00:00:00:00:07;
n1 => r1 : ICMP - Echo request (src=192.168.64.2 dst=140.96.0.2 ttl=8);
r1 box r1 : ARP - Who has 140.32.0.2? Tell 140.32.0.1;
r2 => r1 : ARP - 140.32.0.2 is at 00:00:00:00:00:07;
r1 => r2 : ICMP - Echo request (src=192.168.64.2 dst=140.96.0.2 ttl=7);
r2 box r2 : ARP - Who has 140.16.0.3? Tell 140.16.0.1;
r4 => r2 : ARP - 140.16.0.3 is at 00:00:00:00:00:11;
r2 => r4 : ICMP - Echo request (src=192.168.64.2 dst=140.96.0.2 ttl=6);
r4 box r4 : ARP - Who has 140.58.0.1? Tell 140.16.0.3;
r3 => r4 : ARP - 140.58.0.1 is at 00:00:00:00:00:20;
r4 => r3 : ICMP - Echo request (src=192.168.64.2 dst=140.96.0.2 ttl=5);
r3 box r3 : ARP - Who has 140.96.0.2? Tell 140.96.0.1;
n5 => r3 : ARP - 140.96.0.2 is at 00:00:00:00:00:05;
r3 => n5 : ICMP - Echo request (src=192.168.64.2 dst=140.96.0.2 ttl=4);
n5 => r3 : ICMP - Echo reply (src=140.96.0.2 dst=192.168.64.2 ttl=8);
r3 box r3 : ARP - Who has 140.16.0.1? Tell 140.16.0.2;
r2 => r3 : ARP - 140.16.0.1 is at 00:00:00:00:00:17;
r3 => r2 : ICMP - Echo reply (src=140.96.0.2 dst=192.168.64.2 ttl=7);
r2 => r4 : ICMP - Echo reply (src=140.96.0.2 dst=192.168.64.2 ttl=6);
r4 box r4 : ARP - Who has 90.20.74.33? Tell 90.20.74.34;
r1 => r4 : ARP - 90.20.74.33 is at 00:00:00:00:00:20;
r4 => r1 : ICMP - Echo reply (src=140.96.0.2 dst=192.168.64.2 ttl=5);
r1 => n1 : ICMP - Echo reply (src=140.96.0.2 dst=192.168.64.2 ttl=4);
```

3.2.4. $ python Main.py topo5.txt n1 n6

(Caso com loop)

```
n1 box n1 : ARP - Who has 192.168.64.1? Tell 192.168.64.2;
r1 => n1 : ARP - 192.168.64.1 is at 00:00:00:00:00:07;
n1 => r1 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=8);
r1 box r1 : ARP - Who has 140.32.0.2? Tell 140.32.0.1;
r2 => r1 : ARP - 140.32.0.2 is at 00:00:00:00:00:07;
r1 => r2 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=7);
r2 box r2 : ARP - Who has 140.16.0.2? Tell 140.16.0.1;
r3 => r2 : ARP - 140.16.0.2 is at 00:00:00:00:00:11;
r2 => r3 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=6);
r3 => r2 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=5);
r2 => r3 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=4);
r3 => r2 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=3);
r2 => r3 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=2);
r3 => r2 : ICMP - Echo request (src=192.168.64.2 dst=90.20.74.66 ttl=1);
r2 box r2 : ARP - Who has 140.16.0.3? Tell 140.16.0.1;
r4 => r2 : ARP - 140.16.0.3 is at 00:00:00:00:00:14;
r2 => r4 : ICMP - Time exceeded (src=140.16.0.1 dst=192.168.64.2 ttl=8);
r4 box r4 : ARP - Who has 90.20.74.33? Tell 90.20.74.34;
r1 => r4 : ARP - 90.20.74.33 is at 00:00:00:00:00:20;
r4 => r1 : ICMP - Time exceeded (src=140.16.0.1 dst=192.168.64.2 ttl=7);
r1 => n1 : ICMP - Time exceeded (src=140.16.0.1 dst=192.168.64.2 ttl=6);
```

4. Considerações

Não foram encontradas dificuldades durante o desenvolvimento do trabalho. Para facilitar a validação com os casos dados pelo professor, foi feito um *script* para a validação automática automática destes resultados. O nome do *script* é *test.sh* e se encontra no mesmo diretório dos módulos. Os casos gerados a partir da topologia proposta foram validados manualmente utilizando os diagramas de troca de mensagens gerados pelo *MsGenny*.