Felipe Dalcin Peiter - Simple Net Simulator

This package is an implementation of a final assignment for the course of Computer Networks I.

It consists in simulating a virtual network from a formatted input file.

After parsing the input, you can execute ping commands between nodes.

SETUP

1. Install the package netaddr:

```
pip install netaddr
```

2. Simple execute the setup.py for automatic install dependencies

```
python setup.py install
```

USAGE

1. Check the input format example and documentation in the docs/input_format.txt.
2. Proceed to create a .txt with the nodes, routers and routertables of the scenario you want to simulate.
3. Execute the program, pass the input file that you created and inform the order of communication between the nodes.

```
python net_simulator.py input.txt n1 n2 n3
```

   In this example, the n1 will proceed to perform an ECHO REQUEST to n2, then, n2 will perform an ECHO REQUEST to n3.

4. The result is then printed in the console.

IMPLEMENTATION DETAILS

Inside the simulator, you can find the network folder, where all the objects and data structures are contained.

The main idea on the project was to think of the simplest way of setting up the environment necessary for correctly simulate the communication between two real network devices.

As relying on broadcast messages and the physical layer, the idea was concentrate the parsing of the request mainly on the routers where the nodes are connected. Even if a node wants to send a ArpPacket or a EchoPacket for another node in the same network, the request is sent to the router, then redirected correctly to the desired recipient.

arp_packet.py

The class ArpPacket encapsulates all the necessary and relevant info about the request from one node to another.

The classes ArpRequest and ArpReply are only used for formatting the string output of the class.

echo_packet.py

The class EchoPacket encapsulates all the necessary and relevant info about the request from one node to another.

The classes EchoRequest and EchoReply are only used for formatting the string output of the class.

node.py

Contains the class Node, that it is responsible for storing the Node info.

The node can generate requests and responses for Arp and Echo packets.

Each request is encapsulated in the correspondent class then sent to the gateway, that is the RouterPort where the node is connected, then processed there.

router.py

The core of the simulator, the Router contains several RouterPorts, where we store the MAC and IP Address, besides the connected Nodes or RouterPorts.

Each router receives and parses all the requests, redirecting all of them based on the SRC and DST Ip Address.

The router also changes the SRC_MAC of each request, so we can easily recognized who are sending or redirecting the request and who are receiving.

router_table.py

The router table stores each Router object info about the connected routes.

EXAMPLES

Let's consider the following input file input.txt:

```
#NODE
n1,00:00:00:00:00:01,192.168.0.2/24,192.168.0.1
n2,00:00:00:00:00:02,192.168.0.3/24,192.168.0.1
n3,00:00:00:00:00:03,192.168.1.2/24,192.168.1.1
n4,00:00:00:00:00:04,192.168.1.3/24,192.168.1.1
n5,00:00:00:00:00:07,192.168.4.2/24,192.168.4.1
n6,00:00:00:00:00:08,192.168.4.3/24,192.168.4.1
#ROUTER
r1,3,00:00:00:00:00:05,192.168.0.1/24,00:00:00:00:00:06,192.168.1.1/24,00:00:00:00:00:10,192.168.3.1/30
r2,2,00:00:00:00:00:09,192.168.4.1/24,00:00:00:00:00:11,192.168.3.2/30
#ROUTERTABLE
r1,192.168.0.0/24,0.0.0.0,0
r1,192.168.1.0/24,0.0.0.0,1
r1,192.168.4.0/24,192.168.3.2,2
r2,192.168.0.0/24,192.168.3.1,1
r2,192.168.1.0/24,192.168.3.1,1
r2,192.168.4.0/24,0.0.0.0,0
```

Then we execute the program with:

```
python net_simulator.py input.txt n1 n2 n5
```

The output:

```
n1 box n1 : ARP - Who has 192.168.0.3? Tell 192.168.0.2;
n2 => n1 : ARP - 192.168.0.3 is at 00-00-00-00-00-02;
n1 => n2 : ICMP - Echo request (src=192.168.0.2 dst=192.168.0.3 ttl=8);
n2 => n1 : ICMP - Echo reply (src=192.168.0.3 dst=192.168.0.2 ttl=8);
n2 box n2 : ARP - Who has 192.168.0.1? Tell 192.168.0.3;
r1 => n2 : ARP - 192.168.0.1 is at 00-00-00-00-00-05;
n2 => r1 : ICMP - Echo request (src=192.168.0.3 dst=192.168.4.2 ttl=8);
r1 box r1 : ARP - Who has 192.168.3.2? Tell 192.168.3.1;
r2 => r1 : ARP - 192.168.3.2 is at 00-00-00-00-00-11;
r1 => r2 : ICMP - Echo request (src=192.168.0.3 dst=192.168.4.2 ttl=7);
r2 box r2 : ARP - Who has 192.168.4.2? Tell 192.168.4.1;
  n5 => r2 : ARP - 192.168.4.2 is at 00-00-00-00-00-07;
r2 => n5 : ICMP - Echo request (src=192.168.0.3 dst=192.168.4.2 ttl=6);
n5 => r2 : ICMP - Echo reply (src=192.168.4.2 dst=192.168.0.3 ttl=8);
r2 => r1 : ICMP - Echo reply (src=192.168.4.2 dst=192.168.0.3 ttl=7);
r1 => n2 : ICMP - Echo reply (src=192.168.4.2 dst=192.168.0.3 ttl=6);
```

CONTACT

Please send bug reports, patches, and other feedback to fdpeiter@gmail.com