CAMADA DE REDE – IP

NESTA PRÁTICA, vamos implementar um protótipo de um protocolo compatível com o IPv4. A implementação será capaz de funcionar como *host*, como roteador, ou como ambos simultaneamente. Focaremos no plano de dados, ou seja, não implementaremos algoritmos de roteamento, apenas o encaminhamento com base em uma tabela previamente montada.

Sua implementação deve ser realizada no arquivo `ip.py`, que já veio com um esqueleto em cima do qual você vai construir o seu código. Para ajudar na sua implementação, você pode chamar as funções e usar os valores que já vieram declarados no arquivo `iputils.py`. Pode ser útil, também, consultar a página sobre o formato do datagrama IPv4 e a página sobre o formato das mensagens ICMP na Wikipedia.

Para testar seu código, execute `./autograde.py`. Cada um dos testes vai usar a sua implementação como uma biblioteca, verificando se ela apresenta o comportamento esperado.

Passo 1 — 2 pontos

Implemente os métodos `definir_tabela_encaminhamento` e `_next_hop` da classe `IP`.

Por enquanto, você pode assumir que não acontecem ambiguidades na tabela de encaminhamento, ou seja, que cada endereço IP de destino casa um único CIDR dentre os fornecidos na tabela.

A seu critério, o método `definir_tabela_encaminhamento` pode simplesmente armazenar a tabela para ser usada depois, ou pode transformá-la em alguma outra estrutura de dados que você julgue ser mais adequada ou eficiente.

O método `_next_hop` deve retornar (como uma *string*) o endereço IP do próximo salto necessário para "chegar mais perto" do endereço IP de destino passado como argumento, de acordo com a tabela de encaminhamento configurada. Se nenhuma entrada da tabela de encaminhamento casar, o método não deve retornar nada (ou retornar `None`).

Passo 2 — 2 pontos

Termine a implementação do método `enviar` da classe `IP`. Monte um datagrama IP que contenha como *payload* o segmento TCP fornecido.

Passo 3 — 2 pontos

Melhore a sua implementação do método `_next_hop` da classe `IP`.

Quando o endereço IP de destino casar com mais de um CIDR da tabela, faça o desempate usando a entrada que apresentar o prefixo mais longo.

Passo 4 — 2 pontos

Melhore a implementação do método `__raw_recv` da classe `IP` para que ela consiga tratar adequadamente o campo de TTL quando estiver realizando a função de roteamento.

Decremente o número contido no TTL antes de encaminhar o datagrama para o próximo roteador. Não se esqueça de corrigir o *checksum* do cabeçalho. Se o TTL chegar a zero, descarte o datagrama em vez de encaminhá-lo.

Passo 5 — 2 pontos

Melhore a implementação do método `__raw_recv` da classe `IP` para que a sua implementação permita diagnóstico de rotas (como os gerados pelo utilitário *traceroute*).

Sempre que o TTL chegar a zero, além de descartar o datagrama, gere uma mensagem do tipo ICMP Time exceeded e envie-a de volta ao remetente.

Opcional

O arquivo `exemplo_integracao.py` complementa o do trabalho anterior, colocando a implementação do protocolo IP que você construiu no lugar da implementação do Linux. Caso você tenha feito todos os trabalhos anteriores, tente portá-los para usar a implementação do IP que você acabou de construir.