Redes I – TP - Implementação de pilha de protocolos

DNS

Jonathan Henrique, Rúbio Torres e Igor Miranda

**1 - Camada de Aplicação**

A implementação da camada de aplicação foi feita em Lua, utilizando o framework LÖVE para demonstrar o funcionamento do DNS.

**1.1 Implementação**

Código do Servidor

     O código do servidor fica executando o tempo todo, aguardando a conexão com a camada física. Após receber a mensagem e o IP de origem, ele passa esses parâmetros para a função new\_request. Esta função adiciona uma marcação de horário à requisição, além de enviá-la a função resolve\_dns, que retornará um IP ou nome, e adicionar a requisição a um log.

DNS

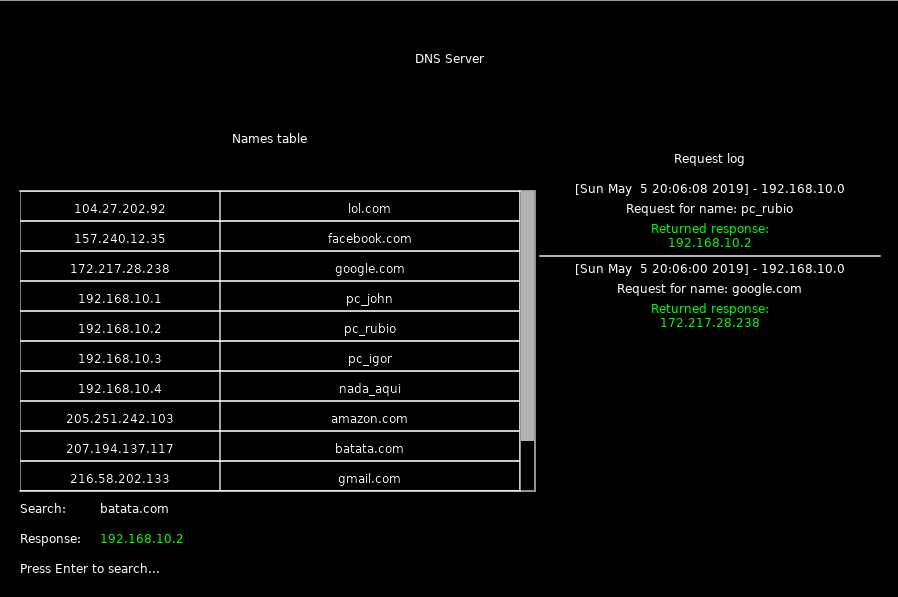
A função resolve\_dns analisa se a requisição é um IP ou nome e realiza a busca em uma tabela já existente no código. Caso seja encontrado nesta tabela o resultado é retornado, caso contrário uma pesquisa do nome ou IP é realizada em um DNS externo, utilizando a função tohostname ou toip existente em lua. Mesmo assim, se a requisição não for encontrada nem no DNS externo, retorna-se nil.

Interface

      A interface foi construída usando as funções do LÖVE, love.draw para desenhar a tela e love.update para controlar as animações, movimentos e o scroll da aplicação e as funções love.keypressed e love.mousepressed para controlar os eventos de input. Há também a função love.load, chamada na inicialização do aplicativo, que inicializa as variáveis para a interface, assim como também faz as chamadas para executar o código do servidor e também inicia automaticamente a camada física (também iniciará as camadas de transporte e redes quando forem implementadas).

**1.2 Testes**

Para testar o funcionamento do servidor, foi desenvolvida uma aplicação em lua utilizando o framework LÖVE, conforme mostra a figura abaixo:



Na tela são exibidos um histórico das requisições, informando uma data, hora, nome solicitado e a resposta, assim como uma tabela dos registros locais de IPs e nomes do servidor DNS. Além disso, ao apertar espaço, é enviada uma solicitação em binário para a camada física, para se fazer um teste mais completo.

**1.3 Execução**

Para executar arquivos love no linux basta abrir o terminal e digitar:

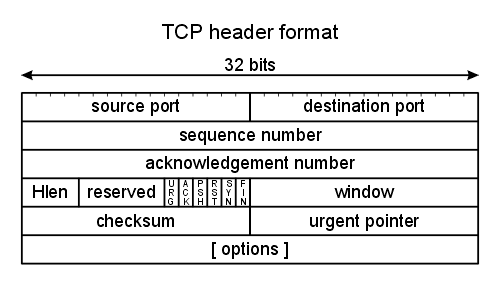
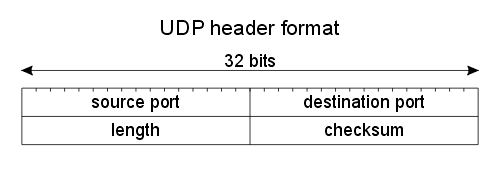
|  |
| --- |
| *sudo add-apt-repository ppa:bartbes/love-stable*  *sudo apt-get update*  *sudo apt-get install love*  *sudo love ApplicationLayer* |

Após isso abrirá a aplicação em love que chamará a camada física em python que rodará em segundo plano a todo momento.

Assim, digitando no search o ip ou o nome do host  a aplicação funciona, tanto como DNS ou DNS reverso e salva na tabela de DNS, mostrada na tela.

**2 - Camada de Transporte**

A implementação da camada de transporte foi feita em PHP, usando a biblioteca de socket. De acordo com a especificação dos protocolos, ela opera com cabeçalhos no seguinte formato:

A camada recebe uma mensagem da camada de Aplicação, e coloca o cabeçalho com todos os dados necessários. Caso o protocolo usado seja UDP, ela envia para a outra camada a mensagem inteira, e, caso o protocolo seja TCP, ela executa o Three-Way Handshake, monta o cabeçalho do protocolo, faz o controle de fluxo e envia para o destinatário a mensagem dividida em pacotes menores. Ao receber uma mensagem de uma camada inferior, ela faz o processo inverso e entrega à camada de aplicação.

**2.1 Implementação**

Loop principal do servidor

Ao executar o servidor, ele fica constantemente “escutando” por conexões. Quando uma máquina solicita uma conexão, ele aceita e espera o recebimento de dados. Ao receber os dados, a máquina verifica a origem para saber se os dados são provenientes da camada de Aplicação ou de uma camada inferior, e toma a ação apropriada.

Envio de mensagens – TCP

Quando a camada recebe um código da camada de Aplicação a ser transferia por TCP, ela, primeiro, inicia um Three-Way Handshake para iniciar a conexão com o destino, se essa conexão já não existia, e, depois, divide a mensagem em pacotes menores e os transmite. Se é solicitada uma retransmissão (3 ACKs consecutivos), é recomeçada a retransmissão a partir do ponto da última mensagem não confirmada pelo cliente. Além disso, o tamanho de cada mensagem é incrementado a cada mensagem recebida com sucesso, e retorna ao valor inicial (10 bytes) quando ocorre algum erro.

Envio de mensagens – UDP

Mensagens UDP são bem mais simplesmente tratadas, o código simplesmente insere os cabeçalhos e passa para a camada inferior.

Recebimento de mensagens – TCP

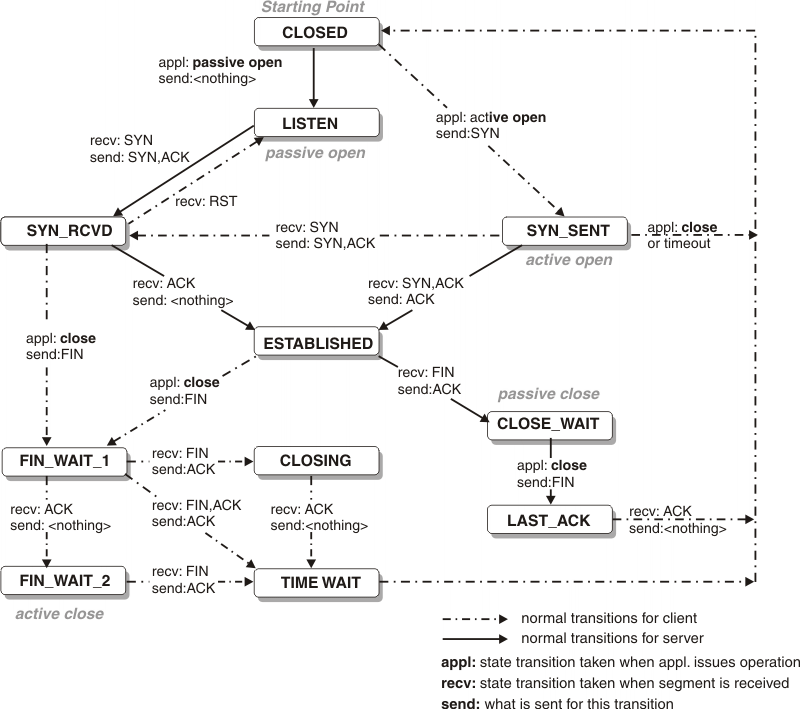
Ao receber segmentos TCP, considerando que já houve o Three-Way Handshake e a conexão já está estabelecida, a camada compara se ela esperava pelo número de sequência recebido, ou seja, se o segmento recebido é o próximo na mensagem que ela está recebendo, e, em caso afirmativo, ela adiciona o segmento à mensagem na memória. Em caso negativo, ela solicita retransmissão, enviando novamente o número de confirmação da mensagem anterior. Após a mensagem inteira ter sido recebida, a camada envia a mensagem completa à camada de Aplicação.

Recebimento de mensagens – UDP

Mensagens UDP recebidas pelo código tem o cabeçalho removido e, após isso, são repassadas para a camada de Aplicação.

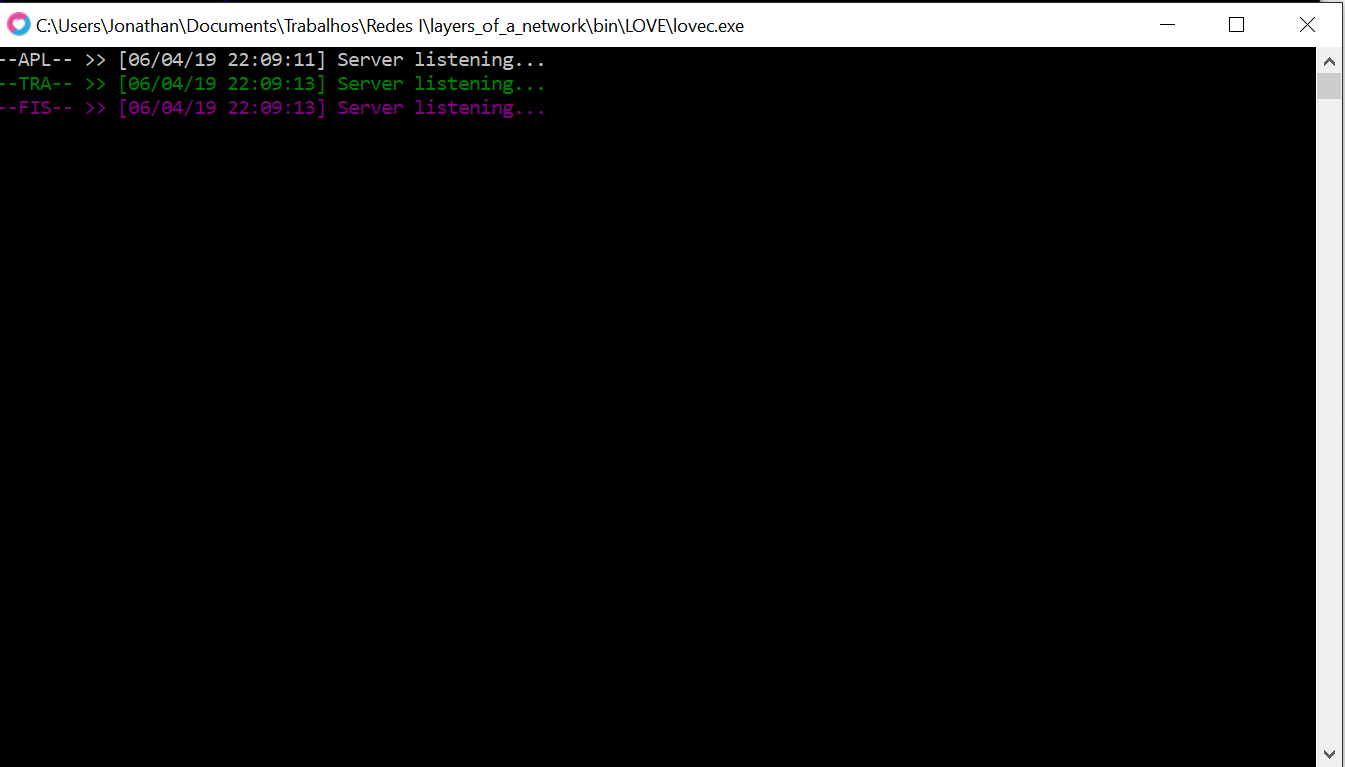
Three-Way Handshake

Ao iniciar ou finalizar uma conexão, o código faz um Three-Way Handshake quando está sendo usado o protocolo TCP. A cada passo desse processo, a camada assume um estado. Esse procedimento foi feito de acordo com o fluxograma a seguir:



**2.2 Testes**

Para testar o funcionamento da camada, ela foi integrada ao código já pronto das camadas de Aplicação e Física. Além disso, os logs de cada camada foram alterados para serem exibidos em cores diferentes, para facilitar a visualização dos logs.



**2.3 Execução**

Para executar o programa, simplesmente entre em um terminal, vá para a pasta “src” e execute “love ApplicationLayer” (é necessário o LÖVE2D para executar o programa). Com a aplicação aberta, simplesmente digite uma consulta e pressione “Enter” para iniciar o envio de uma resposta DNS à requisição digitada. É possível clicar na caixa onde está sendo indicado o protocolo da camada de Transporte para alterá-lo entre TCP e UDP.

**3 - Camada Física**

A implementação da camada física foi feita em Python, usando a biblioteca de socket. De acordo com o RFC, ela opera com PDUs no seguinte formato:



O código recebe um payload de dados da camada superior, e coloca o cabeçalho com todos os dados necessários, os converte em binário, e envia para o destinatário, ou faz o processo inverso ao receber mensagens de fora.

**3.1 Implementação**

Código do Servidor

Ao executar o servidor, ele fica constantemente “escutando” por conexões. Quando uma máquina solicita uma conexão, ele aceita e espera o recebimento de dados. Ao receber os dados, a máquina verifica a origem para saber se os dados são provenientes de uma camada superior ou do ambiente externo, e toma a ação apropriada (monta um quadro para enviar caso seja da camada superior, ou desmonta e decodifica o quadro para mensagens vindas de fora).

Montagem e decodificação de Quadros

Para montar o quadro, o servidor começa, inicialmente, pelo *preamble*, uma sequência de 7 bytes de 1s e 0s alternados, e mais um byte terminando com 11 (10101011) para demarcar o fim desse trecho. Depois, temos o endereço MAC de origem e de destino, codificados em binário. O código recebe o MAC de origem como uma configuração interna, e calcula o MAC de destino a partir do IP. Para fazer esse cálculo, o código confere se o MAC existe na tabela local, e, caso não exista, faz o ARP (via linha de comando), encontra o ARP e registra na tabela local. Depois, ele coloca 2 bytes para representar o tamanho total da mensagem (EtherType). Esse valor pode ter outro significado acima de 1500, mas, como as mensagens de DNS são curtas, esse valor é sempre o tamanho da mensagem. Depois, se segue o binário do payload, convertido de string, e, por final, o FCS (Frame Check Sequence), que é um valor usado para verificação do pacote.

Quando o servidor recebe uma mensagem de fora, ele faz o processo reverso, ou seja, ele separa os bits da mensagem de acordo com o tamanho de cada campo, e faz a conversão desses bits para valores legíveis. O servidor também faz a verificação do FCS, mas, como essa parte não fazia parte da especificação do trabalho, não poderíamos esperar que todas as mensagens dos outros grupos tivessem o FCS implementado, e, portanto, ele aceita as mensagens mesmo quando a verificação falha (mas ainda avisa quando falha).

Envio de Dados

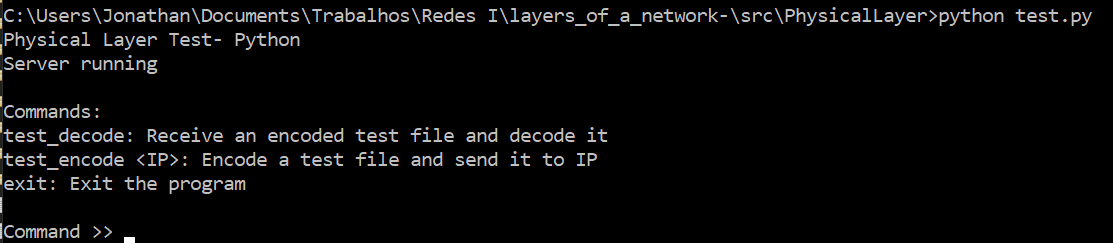
Ao finalizar a montagem de quadro, o servidor simplesmente envia para o destinatário via socket, usando o IP do destinatário, pois, para essa implementação, não seria viável colocar a mensagem diretamente no meio físico. Porém, antes de enviar, o código simula a probabilidade de colisão (5%), e, em caso de colisão, ele espera um tempo aleatório (entre 0.01 e 1 seg) e tenta enviar novamente.

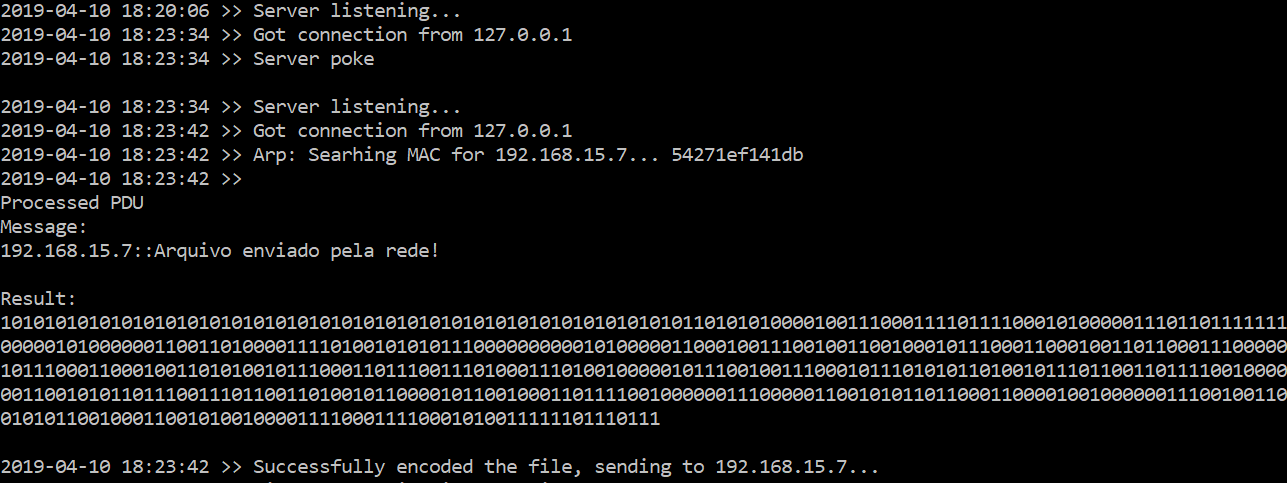
FCS

Como foi mencionado anteriormente, o código faz a verificação de pacotes. Para isso, o código gera um valor baseado no dado do payload, e verifica esses dados quando recebe pacotes externos. Como o código gerador de FCS é um algoritmo usual, foi usado um código da internet para a geração e checagem do valor. O código gera um valor de 32 bits, usando um polinômio gerador “101010101010101010101010101010101” (o polinômio gerador deve ter 1 bit a mais que o resultado desejado)

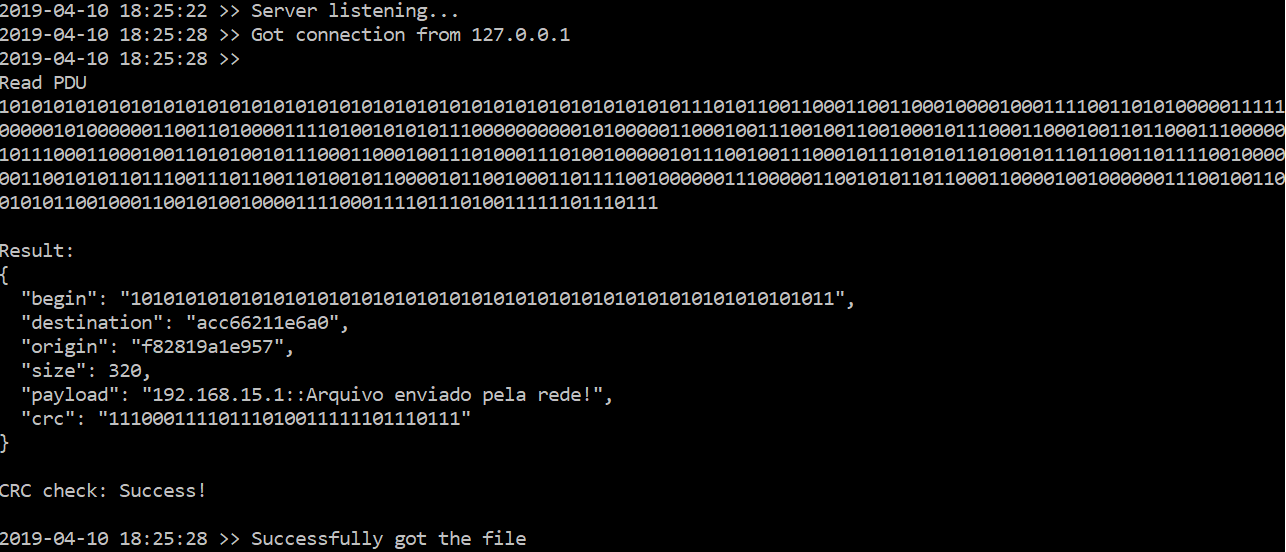
**3.2 Testes**

Para testar o funcionamento do servidor sem as camadas superiores implementadas, foi desenvolvida uma pequena aplicação Python para fazer comunicação com esse servidor e testar as funcionalidades.

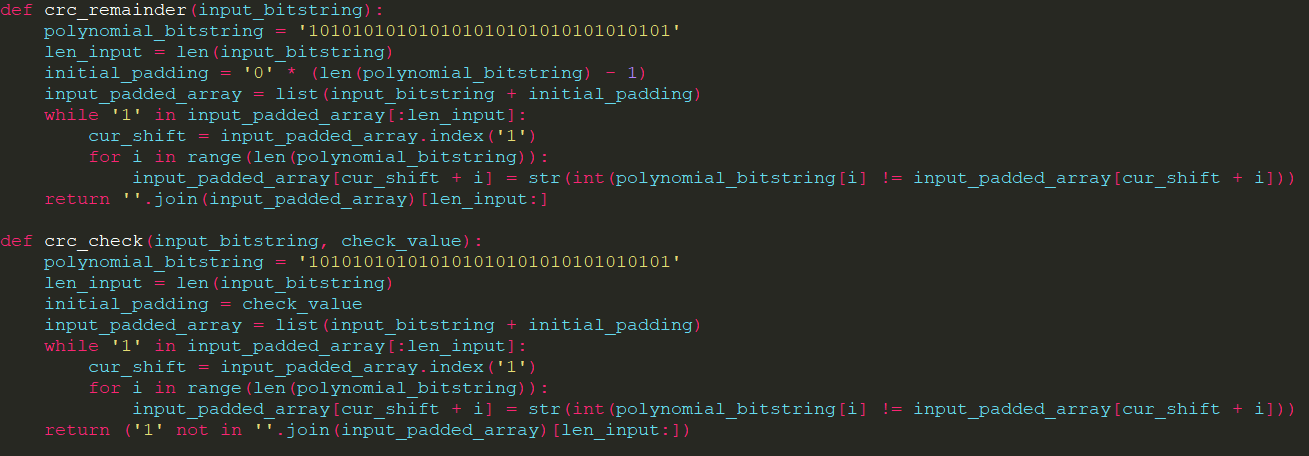


Essa aplicação envia mensagens para o servidor, sendo essa uma mensagem do mundo externo ou de camada superior, além de, na inicialização, enviar uma mensagem de conferência do status do servidor (Uma mensagem cujo conteúdo é somente “Hello”), que o servidor foi programado para responder.

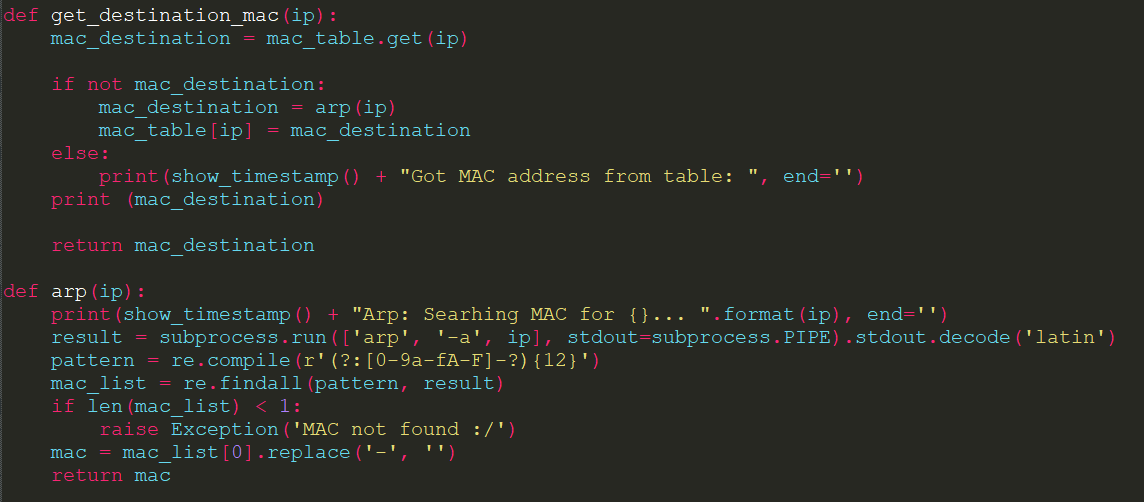
Resposta do servidor ao receber o ping (apelidado “*poke*”) e codificação de mensagem para binário



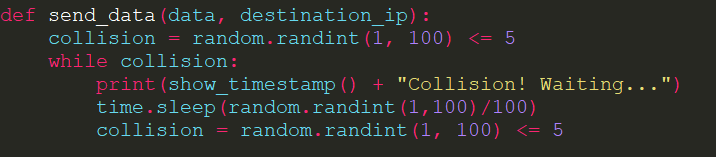
Resposta do servidor ao receber uma mensagem externa, fazendo a decodificação e separando os campos



Código do CRC, disponível em <https://en.wikipedia.org/wiki/Cyclic_redundancy_check> (com modificações)



Código do ARP



Código de colisão



Montagem e “desmontagem” de quadros

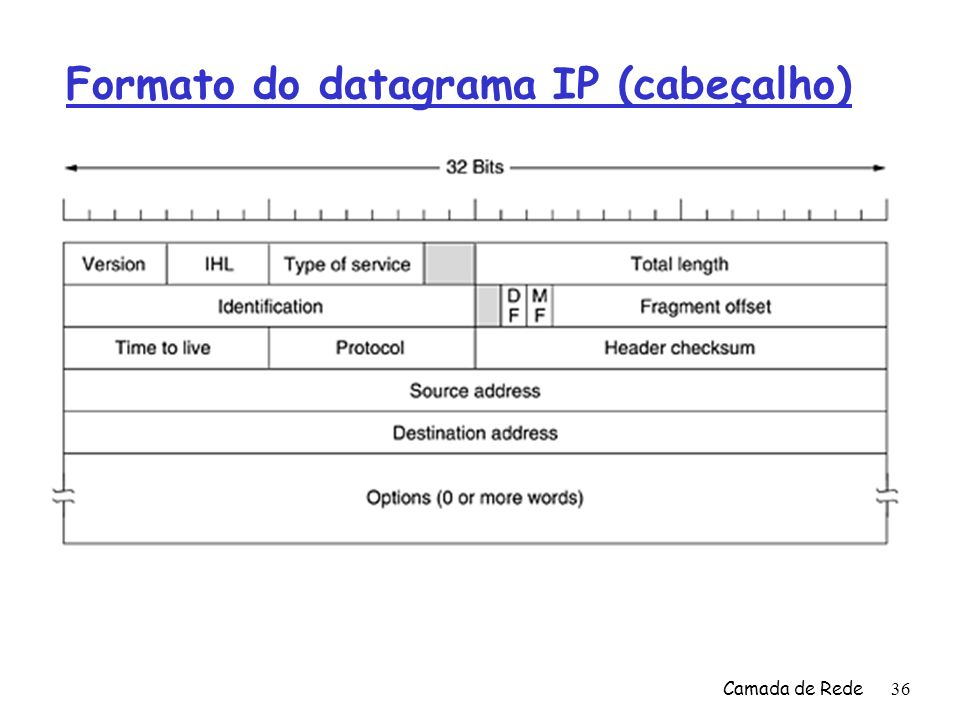
**3.3 Execução**

Para executar o programa, simplesmente entre em um terminal, vá para a pasta do código e execute “python server.py”. Para executar a aplicação de testes, execute “python test.py” em um outro terminal. Para testar a codificação de mensagens, digite na aplicação de testes “test\_encode IP”, sendo IP o endereço da máquina de destino. Para testar a decodificação, basta enviar ao servidor um arquivo codificado, e ele exibirá todo o processo, conforme mostrado nos *prints* acima.

**4- Camada de Rede**

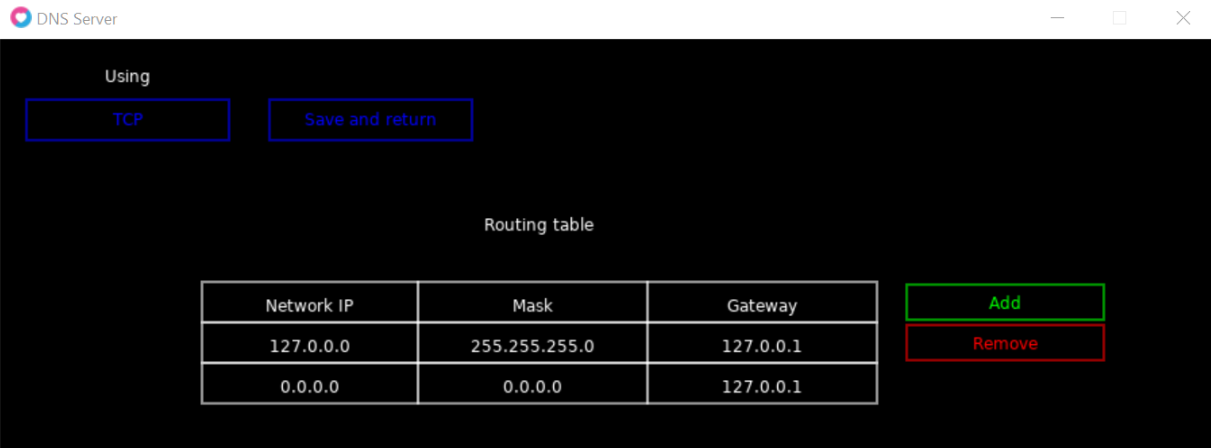
**4.1 Implementação**

A implementação da camada de rede foi feita em Javascript. De acordo com a especificação dos protocolos, ela opera com cabeçalhos no seguinte formato:

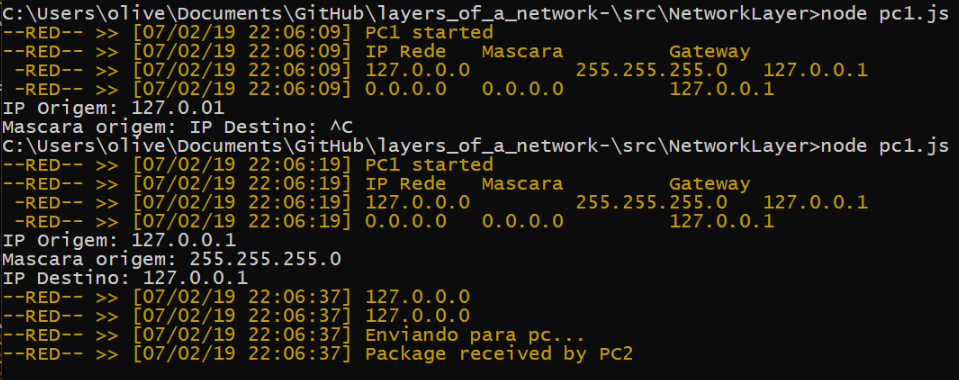


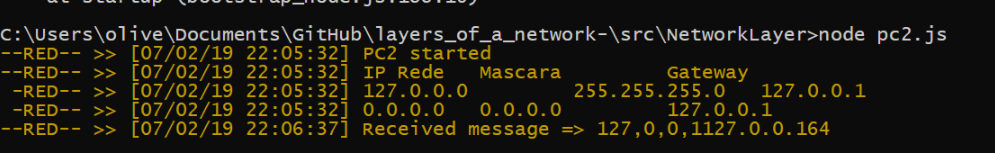
**4.2 Testes**

A partir da interface gráfica da aplicação é possível visualizar a tabela de roteamento, assim como adicionar ou remover suas entradas. A última entrada da tabela foi definida como a default.

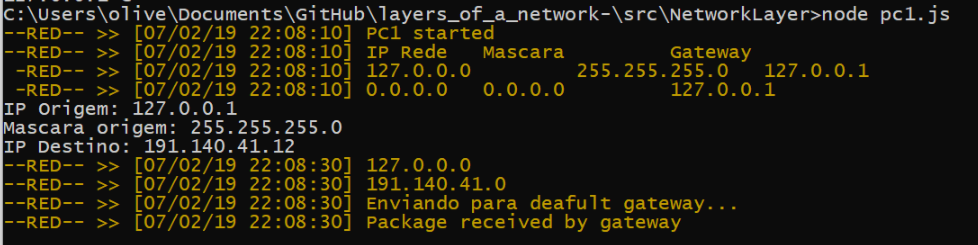


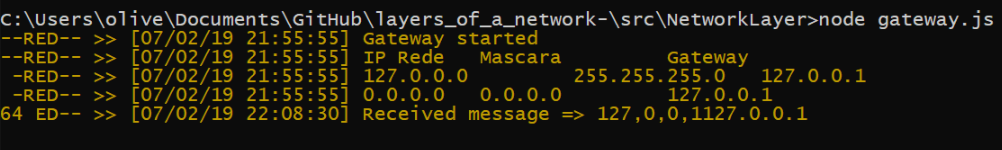
Executando individualmente os códigos da camada de rede é possível enviar uma mensagem de PC1 para PC2, que estão na mesma rede:





Ao enviar uma mensagem de PC1 para um IP que não pertence a rede, a tabela de roteamento é consultada e o envio é realizado. Neste segundo caso quem recebeu a mensagem foi o gateway padrão da rede:





**4.3 Execução**

Para executar a camada de rede é necessário realizar os comandos:

node gateway.js

node PC1.js

node PC2.js

**Considerações Finais**

A camada de aplicação foi implementada com um funcionamento que pode ter pequenas alterações para se adequar à implementação das camadas inferiores, ou para se adequar às implementações dos outros grupos, visando a comunicação mútua. Tudo foi implementado de acordo com a especificação, exceto pela adição da interface gráfica, o uso do DNS externo e de outros atributos que foram implementados a fim de dar uma dinâmica melhor à aplicação.

A camada de transporte foi implementada com um funcionamento que pode ter pequenas alterações para se adequar à implementação da camada de rede. Tudo foi implementado de acordo com a especificação.

A camada física foi implementada com um funcionamento que pode ter pequenas alterações para se adequar à implementação das camadas superiores, ou para se adequar às implementações dos outros grupos, visando a comunicação mútua. Tudo foi implementado de acordo com a especificação, exceto pela adição da conferência do FCS, e pelo fato de que o log do processamento e geração de PDUs é exibido por padrão, sem necessidade de um comando para exibí-los.