

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Departamento de Computação

Redes de Computadores I

Implementação das Camadas

TCP/IP

**Camadas de Aplicação, Transporte, Rede e Física.**

Eduardo Humberto

Felipe Freitas

Mariana Bulgarelli

Yulli Dias

**Sumário**

[**I.** **INTRODUÇÃO** 3](#_Toc532127764)

[**II.** **CAMADAS TCP/IP IMPLEMENTADAS** 4](#_Toc532127765)

[**1.** **Camada de Aplicação** 4](#_Toc532127766)

[**1.2.** **Escopo de Implementação** 6](#_Toc532127767)

[1.2.1. Definição do servidor local 6](#_Toc532127768)

[1.2.2. Definição o protocolo de comunicação entre as camadas físicas 6](#_Toc532127769)

[1.2.3. Definição das funções para os sockets da linguagem utilizada 6](#_Toc532127770)

[1.2.4. Definição das funções de comunicação e conversão 6](#_Toc532127771)

[1.2.5. Definição da codificação do quadro de dados 6](#_Toc532127772)

[1.2.6. Definição do DNS. 6](#_Toc532127773)

[**2.** **Camada de Transporte** 7](#_Toc532127774)

[**2.1** **Descrição** 7](#_Toc532127775)

[**2.2** **Escopo de Implementação** 8](#_Toc532127776)

[2.1.1. Definição do servidor local 8](#_Toc532127777)

[2.1.2. Definição o protocolo de comunicação entre as camadas 8](#_Toc532127778)

[2.1.3. Definição das funções para os sockets da linguagem utilizada 8](#_Toc532127779)

[2.1.4. Definição das funções de comunicação e conversão 8](#_Toc532127780)

[2.1.5. Definição da codificação do quadro de dados 8](#_Toc532127781)

[**3.** **Camada de Rede** 10](#_Toc532127782)

[**4.** **Camada Física** 11](#_Toc532127783)

[**4.1.** **Descrição** 11](#_Toc532127784)

[**4.2.** **Escopo de Implementação** 12](#_Toc532127785)

[4.2.1. Definição do servidor local 12](#_Toc532127786)

[4.2.2. Definição o protocolo de comunicação entre as camadas 12](#_Toc532127787)

[4.2.3. Definição das funções para os sockets da linguagem utilizada 12](#_Toc532127788)

[4.2.4. Definição das funções de comunicação e conversão de formatos 12](#_Toc532127789)

[4.2.5. Definição da codificação do quadro de dados 13](#_Toc532127790)

[4.2.6. Definição do modo de colisão. 13](#_Toc532127791)

[**4.3.** **Códigos** 13](#_Toc532127792)

[**III.** **Modo de execução** 22](#_Toc532127793)

[**IV.** **Resultados** 23](#_Toc532127794)

[**V.** **Referências Bibliográficas** 25](#_Toc532127795)

**Índice de Figuras**

[Figura 1: Esquema de funcionamento. 3](#_Toc532127796)

[Figura 2: Camadas. 4](#_Toc532127797)

[Figura 3: Consultas DNS. 4](#_Toc532127798)

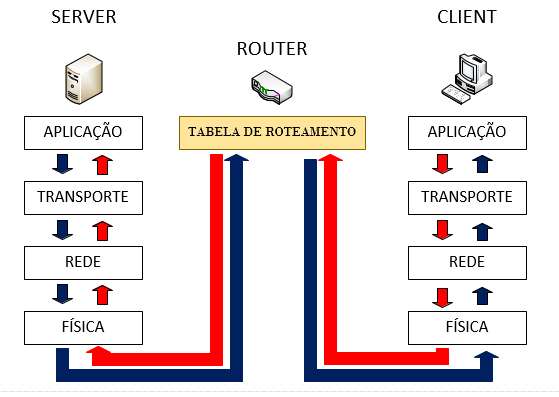
[Figura 4: Cabeçalho TCP. 9](#_Toc532127799)

[Figura 5: Cabeçalho UDP. 10](#_Toc532127800)

# **INTRODUÇÃO**

A mensagem é enviada da camada de Aplicação do Cliente para sua camada de Transporte, onde a mensagem é colocada em um segmento, sendo que pode ser utilizado tanto o protocolo TCP (ralizando o 3 way handshake) quanto o UDP (apenas uma simulação realizada com o socket TCP). Da camada de Transporte, o conjunto segue para a camada de Rede, onde o segmento é posto em um datagrama e segue para a camada Física. Na camada Física, o datagrama da camada de Rede é inserido em um quadro que é enviado para o Roteador. No Roteador há uma tabela que direciona o conteúdo recebido em um enlace para outro enlace, que vai direcionar o mesmo para o servidor. Assim, o Roteador encaminha o conteúdo para a camada física do Servidor. Da camada Física, o conteúdo é retirado do quadro e segue para a camada de Rede. Na camada de Rede, é removido o cabeçalho do datagrama e o conteúdo segue para a camada de Transporte. Nesta camada, é removido o cabeçalho TCP ou UDP (dependendo do caso) e em seguida o conteúdo da requisição segue para sua camada de Aplicação, onde a requisição DNS é respondida e a mensagem segue o fluxo contrário, até chegar a resposta no Cliente (Figura 1).

Figura 1: Esquema de funcionamento.



Fonte: Autores.

A seguir são apresentados detalhes da implementação realizada, sendo que foram utilizadas as reguintes linguagens para as camadas:

Aplicação: Ruby;

Transporte: C#;

Rede: R;

Física: PHP.

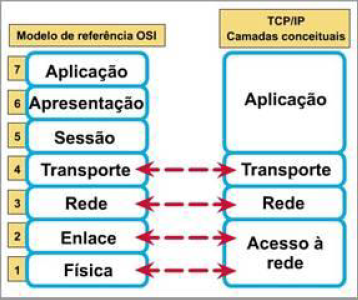
# **CAMADAS TCP/IP IMPLEMENTADAS**

## **Camada de Aplicação**

* 1. **Descrição**

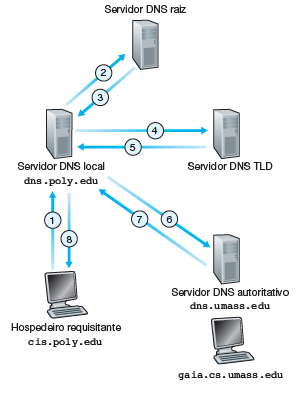
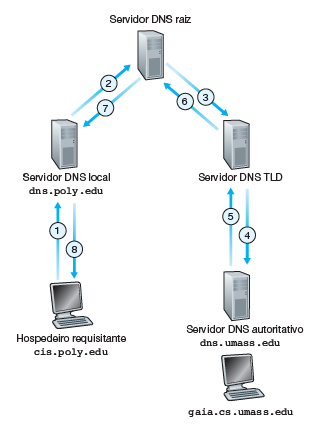
Para a implementação da camada de Aplicação (Figura 2) da pilha de protocolos TCP/IP, foi utilizada a linguagem de programação Ruby. Foi implementado um servidor DNS (Domain Name System), responsável por localizar e traduzir para IP os domínios dos sites que digitamos nos navegadores [1]. As consultas DNS podem ser recursivas ou iterativas (Figura 3).

Figura 2: Camadas.



Fonte: SILVA, 2018 [2].

Figura 3: Consultas DNS.



Fonte: KUROSE, 2013 [3]

Para a implementação dos sockets e implementação dos métodos foi utilizado o conteúdo do Ruby-Doc [4] e RubyDoc.info [5]. A versão do Ruby utilizada foi a 2.5.

Foram consultadas as RFCs 1034 e 1035 (Request for Comments), documentos técnicos desenvolvidos pelo IETF (Internet Engineering Task Force) [6] [7].

Foram definidas portas para a comunicação entre as camadas.

Nesta camada, os parâmetros necessários ao funcionamento são passados via linha de comando quando da execução dos arquivos runServidor.sh e Cliente.rb, que são o localhost e o localhost + domínio/IP, respectivamente.

Foram utilizados Sockets para a comunicação entre as camadas do Servidor e do Cliente e entre Cliente, Router e Servidor e, para tanto, utilizou-se as funções de sockets da linguagem ruby para a camada de Aplicação, em PHP para a camada Física, C# para a camada de Transporte e R para a camada de Rede.

Sockets são terminais de um canal de comunicação bidirecional e podem se comunicar dentro de um processo, entre processos na mesma máquina ou entre máquinas diferentes. A Classe Socket fornece acesso às implementações do socket do sistema operacional subjacente [4].

O processo de comunicação ocorre da seguinte forma:

* Primeiro começa a execução do código do servidor.
* é criado um socket para estabelecer a comunicação. Para tal, utilizou-se a função Socket.new (cria um socket).

socket\_servidor\_aplicacao = Socket.new( AF\_INET, SOCK\_STREAM, 0 )

Onde AF\_INET(IPv4 baseado nos protocolos de Internet. TCP e UDP são protocolos comuns dessa família de protocolos); o tipo de comunicação para ser usado pelo socket foi SOCK\_STREAM.

* Passa-se um nome para o socket por meio da função socket.bind, que passa o nome dado em address para o socket.

Socket.pack\_sockaddr\_in()

socket\_servidor\_aplicacao.bind( )

* É aberta uma escuta para uma conexão no socket por meio da socket.listen. A atividade é registrada no log.

socket\_servidor\_aplicacao.listen( )

* Por meio da função socket.accept, as conexões vindas no socket criado serão aceitas.

socket\_servidor\_aplicacao.accept

* Quando há recebimento de informação, o socket é lido.

recvfrom(maxlen, flags) => [mesg, sender\_addrinfo]

* O quadro é recebido e então a mensagem contida nele é escrita no socket.
* Por fim, a conexão é fechada.

socket\_servidor\_aplicacao.close

* Durante a execução do código do cliente ocorre um processo similar de transmissão e recebimento via sockets.
  1. **Escopo de Implementação**

A implementação foi dividida da seguinte forma:

* + 1. Definição do servidor local

IP do host: 127.0.0.1

* + 1. Definição o protocolo de comunicação entre as camadas físicas

Foi utilizado o protocolo ARP (Address Resolution Protocol).

* + 1. Definição das funções para os sockets da linguagem utilizada

Socket.new() - cria um socket

Socket.bind() - “amarra” o socket ao host e porta definidos

socket.close() - fecha o socket

puts - escreve os bytes no socket

Socket.listen() - “escuta” uma conexão no socket

Socket.accept() - “aceita” uma conexão no socket

* + 1. Definição das funções de comunicação e conversão

time\_stamp () - retorna data e hora atual.

write\_to\_file (file\_name, content) - escreve um evento no log.

descobreIP (enderecoIPperguntado) - retorna o domínio referente a um IP requisitado.

retornaIP (dominioPerguntado) - retorna um IP referente a um domínio requisitado.

* + 1. Definição da codificação do quadro de dados

Na camada Física codificação do quadro foi realizada utilizando como referência o padrão para transmissão de datagramas IP *RFC895*, com algumas alterações. O campo *preâmbulo* foi modificado para 4 bits com o valor “0101” e o campo *CRC* foi modificado para palavra ERRO, codificada em binário.

As RFCs 1034 e 1035 apresentam as definições para a requisição e resposta do DNS.

* + 1. Definição do DNS.

O Servidor DNS recebe a requisição do cliente. Dependendo da requisição do cliente, ele retorna um endereço IP ou um domínio. No caso, implementamos de duas maneiras distintas. Em uma é utilizada a classe Class: Resolv::DNS em ruby que por meio das funções:

[#getaddress(name) ⇒ Object](https://www.rubydoc.info/stdlib/resolv/Resolv/DNS#getaddress-instance_method)

Gets the IP address of name from the DNS resolver.

[#getname(address) ⇒ Object](https://www.rubydoc.info/stdlib/resolv/Resolv/DNS#getname-instance_method)

Gets the hostname for address from the DNS resolver.

Busca no DNS resolver o endereço IP de um domínio e o domínio de um endereço.

A outra maneira implementada, pega a requisição do Cliente e busca em um arquivo .txt o respectivo domínio ou endereço IP. Ambas as maneiras podem ser utilizadas

## **Camada de Transporte**

* 1. **Descrição**

Foi implementada a camada de Transporte da pilha de protocolos TCP/IP, utilizando a linguagem de programação C#. Foram implementados os protocolos UDP e TCP.

O protocolo UDP é um serviço não confiável (não garante a chegada dos dados intactos ao destinatário) e o tráfego não é regulado.

Já o TCP oferece uma transferência confiável de dados, usando controle de fluxo, números de sequência, reconhecimentos e temporizadores. Apresenta também controle de congestionamento, que “evita que qualquer outra conexão TCP abarrote os enlaces e roteadores entre hospedeiros comunicantes com uma quantidade excessiva de tráfego” (KUROSE, 2013).

Para a implementação dos sockets e implementação dos métodos foi utilizado o conteúdo disponibilizado pela Microsoft [8] [9]. Antes foi necessário configurar o ambiente para a execução, instalando os itens necessários. Os procedimentos necessários são detalhados no site da Microsoft [10] [11] [12] [13]. Deve-se atentar para as versões instaladas, pois versões diferentes do .NET Framework podem impedir que o programa funcione corretamente.

Foram consultadas as RFCs 793 e 768 (Request for Comments), documentos técnicos desenvolvidos pelo IETF (Internet Engineering Task Force) [14] [15], referentes aos protocolos TCP e UDP, respectivamente.

Foram utilizados Sockets para a comunicação entre Cliente e Servidor e, para tanto, utilizou-se as funções de sockets da linguagem C#. Sockets são terminais de um canal de comunicação bidirecional e podem se comunicar dentro de um processo, entre processos na mesma máquina ou entre máquinas diferentes.

O processo de comunicação ocorre da seguinte forma:

* Primeiro começa a execução do código do servidor.
* É criado um socket para estabelecer a comunicação.
* Passa-se um nome para o socket.
* É aberta uma escuta para uma conexão no socket.
* As conexões vindas no socket criado serão aceitas.
* Quando há recebimento de informação, o socket é lido.
* Por fim, a conexão é fechada.
* Durante a execução do código do cliente ocorre um processo similar de transmissão e recebimento via sockets.
  1. **Escopo de Implementação**

A implementação foi dividida da seguinte forma:

* + 1. Definição do servidor local

IP do host: 127.0.0.1

* + 1. Definição o protocolo de comunicação entre as camadas

Foram utilizados os protocolos TCP e UDP.

* + 1. Definição das funções para os sockets da linguagem utilizada

Socket listener = new Socket() - cria um socket

listener.Bind() - “amarra” o socket ao host e porta definidos

MeuSocket.Shutdown() - fecha o socket

listener.Listen - escuta uma conexão no socket

MeuSocket.Receive() - recebe uma conexão no socket

MeuSocket.Send() - envia uma conexão no socket

* + 1. Definição das funções de comunicação e conversão

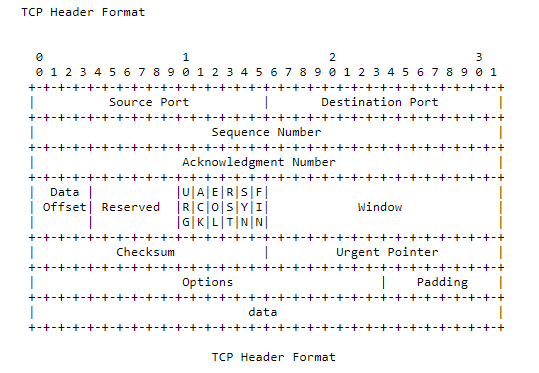
*public static void decodificaSegmentoUDP(string segmento, out string portaOrigem, out string length, out string portaDestino, out string dados)* - Coloca e retira do formato da RFC do protocolo UDP.

*public static void decodificaSegmentoTCP(string pacote, out string portaOrigem, out string portaDestino,out string seq, out string ack, out string offset, out string reserved, out string ackControl, out string synControl, out string finControl, out string rwnd)* - Coloca e retira do formato da RFC do protocolo TCP.

* + 1. Definição da codificação
       1. Recebe mensagem da camada de aplicação.
       2. TCP

1. Monta o pacote com os seguintes campos:

Figura 4: Cabeçalho TCP.



Fonte: RFC 793

1. [Source Port] - número da porta de origem - 16 bits
2. [Destination Port] - número da porta de destino - 16 bits
3. [Sequence Number] - número de sequência (seq) - 32 bits
4. [Acknowledgment Number] - ACK - 32 bits - esse campo contem o valor do próximo número de sequência que o remetente está esperando receber.
5. [Data Offset] - 4 bits - O número de palavras de 32 bits no cabeçalho TCP. Isso indica onde os dados começam. O cabeçalho TCP incluindo opções é um número integral de 32 bits.
6. [Reserved] - 6 bits - reservado para uso futuro - valor fixo: 0.
7. [Control Bits] - 6 bits

URG: 1 bit (0 ou 1) - campo Urgent Pointer significante  
 ACK: 1 bit - campo Acknowledgment significante

EOL: 1 bit (0 ou 1) – Fim.

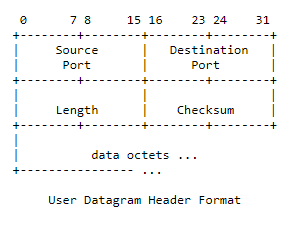
RST: 1 bit (0 ou 1) - Resetar a conexão.

SYN: 1 bit (0 ou 1)

FIN: 1 bit (0 ou 1) - Sem mais dados do transmissor.

1. [Window] - 16 bits - O número de octetos de dados que começam com o indicado no campo de confirmação que o remetente deste segmento está disposto a aceitar.
2. [Checksum] - 16 bits.
3. [Urgent Pointer] - 16 bits - aponta para o número de seqüência do octeto seguindo os dados urgentes e só deve ser interpretado em segmentos com o conjunto de bits de controle URG.
4. [Options] - tamanho variável.
5. [Padding] - tamanho variável - composto por zeros e serve apenas para garantir o tamanho de 32 bits.
6. Three-way Handshake
7. O cliente envia um pacote com a flag SYN ativa (Nº seq 0).
8. Recebe resposta do servidor com as flags SYN e ACK (Nº seq 0, ACK 1) - no pacote de resposta, campo data, deve conter o rwnd (tamanho de buffer disponível).
9. Controle de fluxo
10. Envia pacote com ACK (Nº seq 1)
    * + 1. UDP
11. Monta o pacote com os seguintes campos:

Figura 5: Cabeçalho UDP.



Fonte: RFC 768.

* 1. [Source Port] - Porta de Origem da msgm
  2. [Destination Port] - Porta de Destino da mensagem - 16 bits
  3. [Length] - tamanho do cabeçalho UDP (8 bytes) + tamanho da mensagem em bytes - campo com 16 bits.
  4. [Checksum]
  5. [Data]

## **Camada de Rede**

* 1. **Descrição**

Foi implementada a camada de Rede da pilha de protocolos TCP/IP, utilizando a linguagem de programação R. Foi utilizado o protocolo IP, versão 4: IPv4.

Foi consultada a RFC 791 (Request for Comments), documento técnico desenvolvido pelo IETF (Internet Engineering Task Force) [16], referente ao protocolo IP versão 4, IPv4. O Protocolo da Internet (IP) é um protocolo não confiável e é responsável pelo endereçamento e repasse na Internet.

A camada de rede pega segmentos da camada de transporte e encapsula cada segmento em um datagrama (isto e, em um pacote de camada de rede) ou a camada de rede recebe os datagramas, extrai os segmentos de camada de transporte e os entregara a camada de transporte [3].

O papel primordial dos roteadores e repassar datagramas de enlaces de entrada para enlaces de saída [3].

Para a implementação desta camada utilizou-se o RStudio [17], um software livre de ambiente de desenvolvimento integrado para R.

Para a implementação dos sockets e implementação dos métodos foi utilizado o conteúdo disponibilizado pelo R-Project e R-Bloggers [18] [19].

Foram utilizados Sockets para a comunicação e, para tanto, utilizou-se as funções de sockets da linguagem R.

* 1. **Escopo de Implementação**

Foram implementados Cliente, Roteador e Servidor. O Cliente envia o conteúdo para o Roteador que, através de uma Tabela de Roteamento, direciona o conteúdo recebido do Cliente em um enlace X para o enlace Y, que levará este conteúdo para o Servidor DNC. A implementação foi dividida da seguinte forma:

* + 1. Definição do servidor local

Porta utilizada: 8080

IP do host: 127.0.0.1

* + 1. Definição das funções para os sockets da linguagem utilizada

socket\_create() - cria um socket

socket\_bind() - “amarra” o socket ao host e porta definidos

socket\_close() - fecha o socket

socket\_write() - escreve os bytes no socket

socket\_listen() - “escuta” uma conexão no socket

socket\_accept() - “aceita” uma conexão no socket

* + 1. Definição das funções de comunicação e conversão de formatos

binarioString() - converte um binário para string

stringBinario() - converte uma string para binário

getMac() - retorna o MAC baseado no padrão RFC895, além de verificar a qual MAC cada IP pertence.

macParaBinario() - formata o MAC para um número binário

binarioParaMac() - formata um número binário para um endereço MAC

enviarMensagemServidor() - escreve uma mensagem no socket para o servidor

receberRespostaServidor() - lê o socket e testa se houve resposta do servidor

getMensagemPacote() - retorna a mensagem do pacote, definida em “pacote.txt”

getIpPacote() - retorna o IP do pacote, definido em “pacote.txt”

montaQuadro() - monta um quadro de dados binários

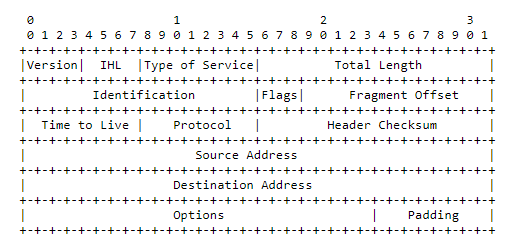
timestamp() - retorna data e hora atual

escreveNoLog() - escreve um evento no log no formato <timestamp><camada : fonte> <evento>

* + 1. Definição da codificação

Foi utilizado o Protocolo IP, IPv4. Para tal, utilizou-se a RFC 791, que define o cabeçalho no seguinte formato:

Figura 6: Cabeçalho IP.



Fonte: RFC 791.

* + 1. [Version]: Número da versão. Especifica a versao do protocolo IP do datagrama. Examinando o numero da versao, o roteador pode determinar como interpretar o restante do datagrama IP [3] - 4 bits.
    2. [IHL]: Comprimento do cabeçalho. Determina onde, no datagrama IP, os dados comecam de fato. A maior parte dos datagramas IP nao contem opções, portanto, o datagrama IP tipico tem um cabecalho de 20 bytes (5 bits) - 4 bits.
    3. [Type of Service]: Tipo de serviço. Os bits de tipo de servico (*type of service* — TOS) foram incluidos no cabecalho do IPv4 para poder diferenciar os diferentes tipos de datagramas. Adotou-se 0000000 - 8 bits.

Bits 0-2: Precedence.

Bit 3: 0 = Normal Delay, 1 = Low Delay.

Bits 4: 0 = Normal Throughput, 1 = High Throughput.

Bits 5: 0 = Normal Relibility, 1 = High Relibility.

Bit 6-7: Reserved for Future Use.

0 1 2 3 4 5 6 7

+-----+-----+-----+-----+-----+-----+-----+-----+

| | | | | | |

| PRECEDENCE | D | T | R | 0 | 0 |

| | | | | | |

+-----+-----+-----+-----+-----+-----+-----+-----+

Precedence

111 - Network Control

110 - Internetwork Control

101 - CRITIC/ECP

100 - Flash Override

011 – Flash

010 – Immediate

001 – Priority

1. - Routine
   * 1. [Total Length]: Comprimento do datagrama. É o comprimento total do datagrama IP (cabecalho mais dados) medido em bytes. Uma vez que esse campo tem 16 bits de comprimento, o tamanho maximo teórico do datagrama IP é 65.535 bytes. Contudo, datagramas raramente sao maiores do que 1.500 bytes [3] - 16 bits
     2. [Identification]: Identificador. Um valor de identificação atribuído pelo remetente para ajudar na montagem dos fragmentos de um datagrama. Adotou-se: 00000000 00000001 -16 bits.
     3. [Flags]: Flags de controle. Adotou-se: 010 - 3 bits

Bit 0: reserved, must be zero.

Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.

Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

0 1 2

+---+---+---+

| | D | M |

| 0 | F | F |

+---+---+---+

* + 1. [Fragment Offset]: Deslocamento de fragmentação*.* Este campo indica em que lugar do datagrama este fragmento pertence. O deslocamento do fragmento é medido em unidades de 8 octetos (64 bits). O primeiro fragmento tem offset zero. Adotou-se: 00000000 00000 - 13 bits. Descartou-se a fragmentação.
    2. [Time to Live]: Tempo de vida. O campo de tempo de vida (time-to-live — TTL) é incluido para garantir que datagramas não fiquem circulando para sempre na rede. Esse campo é decrementado de 1 cada vez que o datagrama é processado por um roteador. Se o campo TTL chegar a 0, o datagrama deve ser descartado [3]. Este procedimento não foi implementado, portanto adotou-se um valor fixo de 00000011 - 8 bits
    3. [Protocol]: Protocolo. O valor do campo indica o protocolo de camada de transporte especifico ao qual a porcao de dados desse datagrama IP deverá ser passada. Por exemplo, um valor 6 indica que a porção de dados será passada ao TCP, enquanto um valor 17 indica que os dados serão passados ao UDP. Adotou-se: 00000110 - TCP - 6 [20] - (Afinal, o socket é TCP sempre - o UDP é uma simulação do TCP em nossa implementação. Como decisão de implementação, mantivemos este valor fixo. 8 bits
    4. [Header Checksum]: Soma de verificação do cabeçalho. A soma de verificação do cabeçalho auxilia um roteador na detecção de erros de bits em um datagrama IP recebido. É calculada tratando cada 2 bytes do cabecalho como se fossem um número e somando esses números usando complementos aritméticos de 1. Soma de verificação não implementada, portanto adotou-se o valor fixo como 00000000 00000000 - 16 bits.
    5. [Source Address]: Endereço IP de Origem. 32 bits.
    6. [Destination Address]: Endereço IP de Destino. 32 bits.
    7. [Options]: Opcoes. Permite que um cabeçalho IP seja estendido. A intenção é que as opções de cabeçalho sejam usadas raramente [3]. Como decisão de implementação não foi utilizado este campo. Adotou-se: 00000000 - variável.
    8. [Padding]: Preenchimento. Usado para garantir que o cabeçalho da Internet termine em 32 bits. O preenchimento é zero. Adotou-se: 00000000 00000000 00000000 - variável.
    9. [Data]: Dados (carga util).

## **Camada Física**

**4.1. Descrição**

Foi implementada a camada física da pilha de protocolos TCP/IP, utilizando a linguagem de programação PHP. Ela é responsável pelo endereçamento e tradução de nomes e endereços lógicos em endereços físicos. Além disso, determina a rota que os dados seguirão do computador de origem até o de destino.

O processo ocorre da seguinte forma (as funções de socket utilizadas foram obtidas no PHP Manual):

* Primeiro começa a execução do código do servidor.
* SERVIDOR: é criado um socket para estabelecer a comunicação. Para tal, utilizou-se a função socket\_create (cria um socket).

resource socket\_create ( int $domain , int $type , int $protocol );

Onde $domain especifica a família do protocolo para ser usado pelo socket, que no caso foi AF\_INET (IPv4 baseado nos protocolos de Internet. TCP e UDP são protocolos comuns dessa família de protocolos); $type seleciona o tipo de comunicação para ser usado pelo socket, que no caso foi SOCK\_STREAM Fornece sequencial, seguro, e em ambos os sentidos, conexões baseadas em "byte streams". O protocolo TCP é baseado neste tipo de socket (PHP MANUAL)). A atividade é registrada em um log geral.

* SERVIDOR: Passa-se um nome para o socket por meio da função socket\_bind (retorna TRUE para sucesso ou FALSE para falha), que passa o nome dado em address para o socket (no caso, atribuímos o nome $socket).

bool socket\_bind ( resource $socket , string $address [, int $port ] );

O parâmetro address passa o endereço IP na notação "dotted-quad" definido pela variável global $MEU\_IP com o valor de “127.0.0.1”. O parâmetro port determina a porta no host remoto para o qual a conexão deve ser feita, que no caso é $MINHA\_PORTA com o valor de “8080”. A atividade é registrada no log.

* Em seguida, o código entra em uma estrutura de repetição do-while que é repetida enquanto a variável $spawn for diferente de falsa. É aberta uma escuta para uma conexão no socket por meio da socket\_listen (retorna TRUE para sucesso ou FALSE para falha, gravado na variável $result). A atividade é registrada no log.

bool socket\_listen ( resource $socket);

* Por meio da função socket\_accept, as conexões vindas no socket criado serão aceitas.

resource socket\_accept ( resource $socket );

* Quando há recebimento de informação, o socket é lido com a função socket\_read () e armazenado em $quadro. O número máximo de bytes lidos é especificado.

string socket\_read ( resource $spawn , intval($TAM\_MAX\_BYTES) );

* O quadro é recebido e então a mensagem contida nele é escrita no socket por meio da socket\_write.

int socket\_write(resource $spawn, string $quadro, int strlen($quadro))

* Durante a execução do código do cliente:
* CLIENTE: é montado o quadro/mensagem que será enviado (em binário), utilizando um array contendo o *MAC* e *IP* definidos previamente.
* São lidos os *IP* de destino e a mensagem do arquivo "pacote.txt"; inicia-se o parâmetro do método de transmissão *RFC*, o *preambulo*; inicia-se o delimitador de início do quadro; é feita a conversão do *MAC* de origem e destino; inicia-se o tipo de *IP*; converte a mensagem para binário; seta o parâmetro de erro, o *CRC* e retorna o quadro no seguinte formato: <*preambulo*.*delimitador*.*MAC\_ORIGEM*.*MAC\_DESTINO*.*tipo*.*mensagem*.*CRC*>.
* Similar ao servidor, é criado um socket, esse socket é conectado ao *IP* e porta de destino.
* O quadro/mensagem é escrito no socket.
* Em seguida o socket é lido (pode ou não haver resposta).
* Por fim, a conexão é fechada.

**4.2. Escopo de Implementação**

A implementação foi dividida da seguinte forma:

* + 1. Definição do servidor local

Porta utilizada: 8080

IP do host: 127.0.0.1

* + 1. Definição o protocolo de comunicação entre as camadas

Foi utilizado o protocolo ARP (Address Resolution Protocol).

* + 1. Definição das funções para os sockets da linguagem utilizada

socket\_create() - cria um socket

socket\_bind() - “amarra” o socket ao host e porta definidos

socket\_close() - fecha o socket

socket\_write() - escreve os bytes no socket

socket\_listen() - “escuta” uma conexão no socket

socket\_accept() - “aceita” uma conexão no socket

* + 1. Definição das funções de comunicação e conversão de formatos

binarioString() - converte um binário para string

stringBinario() - converte uma string para binário

getMac() - retorna o MAC baseado no padrão RFC895, além de verificar a qual MAC cada IP pertence.

macParaBinario() - formata o MAC para um número binário

binarioParaMac() - formata um número binário para um endereço MAC

enviarMensagemServidor() - escreve uma mensagem no socket para o servidor

receberRespostaServidor() - lê o socket e testa se houve resposta do servidor

getMensagemPacote() - retorna a mensagem do pacote, definida em “pacote.txt”

getIpPacote() - retorna o IP do pacote, definido em “pacote.txt”

montaQuadro() - monta um quadro de dados binários

timestamp() - retorna data e hora atual

escreveNoLog() - escreve um evento no log no formato <timestamp><camada : fonte> <evento>

* + 1. Definição da codificação do quadro de dados

A codificação do quadro foi realizada utilizando como referência o padrão para transmissão de datagramas IP *RFC895*, com algumas alterações. O campo *preâmbulo* foi modificado para 4 bits com o valor “0101” e o campo *CRC* foi modificado para palavra ERRO, codificada em binário.

* + 1. Definição do modo de colisão.

A probabilidade de uma colisão foi implementada tanto no cliente quanto no servidor, ou seja, a cada envio de PDU de um lado para outro.

Foi definido um percentual de colisão e criado um vetor de dez posições. Esse vetor foi preenchido de maneira aleatória com zeros e uns, sendo que zero significa que não houve colisão e um significa a ocorrência de colisão no envio. Quando, durante o preenchimento, a quantidade de uns for equivalente ao percentual de colisões definido [dentro das dez posições], as demais posições são preenchidas com zeros. Assim, a probabilidade de ter uma colisão será de x em 10.

Após este procedimento, é sorteado, de maneira aleatória um número significando a posição do vetor [um número de 0 a 9]. Realiza-se a conferência do valor na referida posição do vetor: se for igual a zero, a transmissão é feita normalmente; caso contrário, conta-se um número aleatório de segundos e é realizado um novo sorteio de posição, ou seja, é gerado um número aleatório correspondente a uma posição do vetor que, se contiver o valor um, considera-se que houve colisão e espera um tempo aleatório para depois re-enviar o quadro.

A cada colisão, são acrescentados dois segundos o tempo de espera. Quando o número de tentativas exceder 10, o processo para.

**4.3. Códigos**

*Servidor*

<?php

|  |  |
| --- | --- |
|  | $ARQUIVO\_LOG = "../../log.txt"; |
|  | $MEU\_IP = "127.0.0.1"; |
|  | $MINHA\_PORTA = 8080; |
|  | $TAM\_MAX\_BYTES = '3000000'; |
|  | function binarioParaString($sequenciaDeBits){ |
|  | $string = ''; |
|  | for($i=0; $i<(strlen($sequenciaDeBits)-1); $i+=8){ |
|  | $hex = base\_convert(substr($sequenciaDeBits, $i, 8), 2, 16); |
|  | while(strlen($hex)<2){ |
|  | $hex = '0'.$hex; |
|  | } |
|  | $caracter = pack('H\*', $hex); |
|  | $string .= $caracter; |
|  | } |
|  | return $string; |
|  | } |
|  | function stringParaBinario($string){ |
|  | $stringEmBinario = ''; |
|  | $arrayDeCaracter = str\_split($string); |
|  | foreach($arrayDeCaracter as $caracter){ |
|  | $caracterEmHexadecimal = unpack('H\*', $caracter); |
|  | $caracterEmBinario = base\_convert($caracterEmHexadecimal[1], 16, 2); |
|  | while(strlen($caracterEmBinario)<8){ $caracterEmBinario = '0'.$caracterEmBinario; } //garante que tem 8 bits |
|  | $stringEmBinario .= $caracterEmBinario; |
|  | } |
|  | return $stringEmBinario; |
|  | } |
|  |  |
|  | function obterMenssagemDoQuadro($quadro){ |
|  | $preambulo = substr($quadro, 0, 4); //4 bits |
|  | $sfd = substr($quadro, 4, 8); //8 bits |
|  | $mac\_org = substr($quadro, 12, 48); |
|  | $mac\_dest = substr($quadro, 60, 48); |
|  | $tipo = substr($quadro, 108, 16); //16 bits |
|  | $tam\_dado = strlen($quadro) - 156; //tamanho total - cabeçalho - crc |
|  | $data = substr($quadro, 124, $tam\_dado); |
|  | $data = binarioParaString($data); //converte o pacote para string |
|  | $crc = substr($quadro, 124+$tam\_dado, 32); //crc tem 32 bits |
|  | return $data; |
|  | } |
|  | function timestamp(){ |
|  | $now = getdate(); |
|  | $data = $now['mday'] . ' ' . $now['month'] . ' ' . $now['year'] . ' ' . $now['hours'] . ':' . $now['minutes'] . ':' . $now['seconds'] ." "; |
|  | return $data; |
|  | } |
|  | function escreveNoLog($mensagem){ |
|  | file\_put\_contents ( $GLOBALS['ARQUIVO\_LOG'], timestamp() ."[Física: Servidor] " . $mensagem . ". \n", FILE\_APPEND | LOCK\_EX); //lock\_ex lock exclusivo |
|  |  |
|  | } |
|  | set\_time\_limit(0); //sem timeout |
|  | $socket = socket\_create(AF\_INET, SOCK\_STREAM, 0); |
|  | if($socket === FALSE){ |
|  | escreveNoLog("Socket com a camada física não criado"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Socket com a camada física criado"); |
|  | } |
|  |  |
|  | if(socket\_bind($socket, $GLOBALS['MEU\_IP'], $GLOBALS['MINHA\_PORTA']) === FALSE){ |
|  | escreveNoLog("Erro ao vincular nome para o socket"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Vinculando um nome para o socket"); |
|  | } |
|  |  |
|  | do{ |
|  | $result = socket\_listen($socket); |
|  | if($result === false){ |
|  | escreveNoLog("Errro ao ouvir conexão"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Ouvindo a conexão"); |
|  | } |
|  | $spawn = socket\_accept($socket); |
|  | if($spawn === false){ |
|  | escreveNoLog("Conexão não aceita"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Conexão aceita"); |
|  | } |
|  | $quadro = socket\_read($spawn, intval($TAM\_MAX\_BYTES)); |
|  | if($quadro === FALSE){ |
|  | escreveNoLog("Erro ao receber o quadro"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Quadro recebido"); |
|  | } |
|  |  |
|  | $quadro = trim($quadro); |
|  | $mensagem = binarioParaString($quadro); |
|  | if(strcmp($mensagem, "TAM") == 0){ |
|  | escreveNoLog("Enviando limite máximo"); |
|  | $resposta = stringParaBinario($TAM\_MAX\_BYTES); |
|  | socket\_write($spawn, $resposta, strlen ($resposta)); |
|  | } |
|  | else{ |
|  | escreveNoLog("Mensagem {" .obterMenssagemDoQuadro($quadro) ."} recebida"); |
|  | socket\_write($spawn, $quadro, strlen ($quadro)); |
|  | } |
|  | }while ($spawn != FALSE); |
|  |  |
|  | socket\_close($spawn); |
|  | escreveNoLog("Conexão encerrada"); |
|  | ?> |

*Cliente*

<?php

|  |  |
| --- | --- |
|  |  |
|  | $IP\_ORIGEM = "127.0.0.1"; |
|  | $IP\_DESTINO = "127.0.0.1"; |
|  | $PORTA\_SERVIDOR\_FISICA = 8080; |
|  | $ARQUIVO\_LOG = "../../log.txt"; |
|  | $LIMITE\_MAXIMO\_MENSAGEM = '1024'; |
|  | $MAC\_from\_IP = array( "127.0.0.1" => "d0:df:9a:c4:07:ab"); |
|  |  |
|  | function getMAC($ip, &$macIp){ |
|  | if(array\_key\_exists($ip,$macIp)) |
|  | { |
|  | $mac = $macIp[$ip]; |
|  | } |
|  | else{ |
|  | do{ |
|  | $arp\_scan = shell\_exec("arp-scan " . $ip); //necessario executar como root |
|  | $linhas = explode("\n", $arp\_scan); |
|  | $array = str\_split($linhas[2]); |
|  | $mac = ''; |
|  | $i = 13; |
|  | while($i < strlen($linhas[2]) && $i <=29){ |
|  | $mac = $mac . $array[$i]; |
|  | $i++; |
|  | } |
|  | }while(strlen($mac) < 17); |
|  | $macIp[$ip] = $mac; |
|  | } |
|  | escreveNoLog("Protocolo ARP o ip " . $ip . " pertence ao MAC " . $mac); |
|  | return $mac; |
|  | } |
|  |  |
|  | function macParaBinario($mac){ |
|  | $binario = ''; |
|  | $macArray = explode(':', $mac); |
|  | foreach ($macArray as $hexaComDoisDigitos){ |
|  | $bin = base\_convert($hexaComDoisDigitos, 16, 2); |
|  | while( strlen($bin) < 8){ |
|  | $bin = '0'. $bin; |
|  | } |
|  | $binario = $binario . $bin; |
|  | } |
|  | return $binario; |
|  | } |
|  |  |
|  | function binarioParaMac($binario){ |
|  | $macDesformatado = base\_convert($binario, 2, 16); |
|  | $mac = substr($macDesformatado, 0, 2); |
|  | for ($i = 2; $i < strlen($macDesformatado); $i += 2){ |
|  | $mac = $mac . ":" . substr($macDesformatado, $i, 2); |
|  | } |
|  | return $mac; |
|  | } |
|  |  |
|  | function enviarMessagemServidor($socket, $mensagem){ |
|  |  |
|  | if (socket\_write($socket, $mensagem, strlen($mensagem)) === FALSE) //retorna 0 quando os bits são escritos o operador === é usando para garantir que retornou falso e não 0 |
|  | { |
|  | escreveNoLog("Mensagem não enviada"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Mensagem enviada"); |
|  | } |
|  | } |
|  |  |
|  | function receberRespostaServidor($socket, $limiteMensagem){ |
|  | $resposta = socket\_read ($socket, intval($limiteMensagem)); |
|  | if( $resposta === FALSE){ |
|  | escreveNoLog("Resposta não recebida"); |
|  | return null; |
|  | } |
|  | else |
|  | { |
|  | escreveNoLog("Resposta recebida"); |
|  | return $resposta; |
|  | } |
|  | } |
|  | function timestamp(){ |
|  | $now = getdate(); |
|  | $data = $now['mday'] . ' ' . $now['month'] . ' ' . $now['year'] . ' ' . $now['hours'] . ':' . $now['minutes'] . ':' . $now['seconds'] ." "; |
|  | return $data; |
|  | } |
|  | function escreveNoLog($mensagem){ |
|  | file\_put\_contents ( $GLOBALS['ARQUIVO\_LOG'], timestamp() ."[Física: Cliente] " . $mensagem . ". \n", FILE\_APPEND | LOCK\_EX); //lock\_ex lock exclusivo |
|  |  |
|  | } |
|  | function enviarMensagemEObterRespostaDoServidor($mensagem, $limite){ |
|  | $socket = socket\_create(AF\_INET, SOCK\_STREAM, 0); |
|  | if ($socket === FALSE){ |
|  | escreveNoLog("Socket com a camada física não criado"); |
|  | } |
|  | else{ |
|  | escreveNoLog("Socket com a camada física criado"); |
|  | } |
|  | $result = socket\_connect($socket, $GLOBALS['IP\_DESTINO'], $GLOBALS['PORTA\_SERVIDOR\_FISICA']); |
|  | if($result === FALSE){ |
|  | escreveNoLog("Conexão criada com a camada física"); |
|  | } |
|  | enviarMessagemServidor($socket, $mensagem); |
|  | $resposta = receberRespostaServidor($socket, $limite); |
|  | socket\_close($socket); |
|  | return $resposta; |
|  | } |
|  | function stringParaBinario($string){ |
|  | $stringEmBinario = ''; |
|  | $arrayDeCaracter = str\_split($string); |
|  | foreach($arrayDeCaracter as $caracter){ |
|  | $caracterEmHexadecimal = unpack('H\*', $caracter); |
|  | $caracterEmBinario = base\_convert($caracterEmHexadecimal[1], 16, 2); |
|  | while(strlen($caracterEmBinario)<8){ $caracterEmBinario = '0'.$caracterEmBinario; } //garante que tem 8 bits |
|  | $stringEmBinario .= $caracterEmBinario; |
|  | } |
|  | return $stringEmBinario; |
|  | } |
|  | function binarioParaString($sequenciaDeBits){ |
|  | $string = ''; |
|  | for($i=0; $i<(strlen($sequenciaDeBits)-1); $i+=8){ |
|  | $hex = base\_convert(substr($sequenciaDeBits, $i, 8), 2, 16); |
|  | while(strlen($hex)<2) |
|  | { |
|  | $hex = '0'.$hex; |
|  | } |
|  | $caracter = pack('H\*', $hex); |
|  | $string .= $caracter; |
|  | } |
|  | return $string; |
|  | } |
|  | function getMensagemPacote(){ |
|  | $conteudo = file('../pacote.txt'); |
|  | $split = explode(' ', $conteudo[0]); |
|  | return $split[1]; |
|  | } |
|  | function getIpPacote(){ |
|  | $conteudo = file('../pacote.txt'); |
|  | $split = explode(' ', $conteudo[0]); |
|  | return $split[0]; |
|  | } |
|  | function montaQuadro(&$macIp){ |
|  | $ipDestino = getIpPacote(); |
|  | $mensagem = getMensagemPacote(); |
|  | $preambulo = '0101'; |
|  | $sfd = '10101011'; // Delimitador de início de quadro |
|  | $macOrigem = macParaBinario(getMAC($GLOBALS['IP\_ORIGEM'], $macIp)); |
|  | $macDestino = macParaBinario(getMAC($ipDestino, $macIp)); |
|  | $tipo = '0100100101010000';//IP |
|  | $data = stringParaBinario($mensagem); |
|  | $crc = '01000101010100100101001001001111'; //string ERRO |
|  | return $preambulo.$sfd.$macOrigem.$macDestino.$tipo.$data.$crc; |
|  | } |
|  |  |
|  | $quadro = montaQuadro($MAC\_from\_IP); |
|  |  |
|  | $tamMensagemEmBinario = enviarMensagemEObterRespostaDoServidor(stringParaBinario("TAM"), $GLOBALS['LIMITE\_MAXIMO\_MENSAGEM']); |
|  | $GLOBALS['LIMITE\_MAXIMO\_MENSAGEM'] = binarioParaString($tamMensagemEmBinario); |
|  | print "\n\nlimite " . $GLOBALS['LIMITE\_MAXIMO\_MENSAGEM'] . "\n\n"; |
|  | $N\_maxTentativas = 10; |
|  | $tentativa = 0; |
|  | $a = array\_fill(0, 10, 'null'); |
|  | //print\_r($a); |
|  | $probcolisao = (20\*10)/100;//probabilidade de 20% |
|  | $minrange = 0; |
|  | $maxrange = 1; |
|  | for($w = 0 ; $w < 10; $w ++) { |
|  | $contador = 0; |
|  | for($j=0; $j <= $w; $j ++){ |
|  | if($a[$j] === 1){ |
|  | $contador ++; |
|  | } |
|  | } |
|  | if($contador < $probcolisao) { |
|  | $a[$w] = random\_int($minrange, $maxrange); |
|  | } |
|  | else{ |
|  | $a[$w] = 0; |
|  | } |
|  | } |
|  | $conta = 2; |
|  | while($tentativa < $N\_maxTentativas) { |
|  | $sorteio = random\_int(0, 9); |
|  | //print\_r($sorteio); |
|  | if($a[$sorteio] === 1) { |
|  | $tentativa += 1; |
|  | echo "\nCOLISAO! --- Contagem aleatoria de tempo para tentar outra vez... \n"; |
|  | escreveNoLog("Colisão! Tentativa " . $tentativa); |
|  | sleep($conta); |
|  | $conta = $conta + 2;//incrementa a contagem dos segundos ate tentar reenviar |
|  | } |
|  | else{ |
|  | $tentativa = 0; |
|  | $mensagem = montaQuadro($MAC\_from\_IP); |
|  | $resposta = enviarMensagemEObterRespostaDoServidor($mensagem, $GLOBALS['LIMITE\_MAXIMO\_MENSAGEM']); |
|  | if(strcmp($resposta, $mensagem) == 0){ |
|  | print "\n\nPacote recebido com sucesso!\n\n"; |
|  | } |
|  | break; |
|  | } |
|  | sleep(1); |
|  | } |
|  | if($tentativa == $N\_maxTentativas){ |
|  | escreveNoLog("Número máximo de tentativas para enviar o pacote foi atingido"); |
|  | } |

# **Modo de execução**

1. **Camada de Aplicação**
   1. Instale o Ruby e o PHP para a execução.
   2. Execute o arquivo runServidor.sh, em Servidor.

No terminal, digite:

chmod +x runServidor.sh

sudo ./runServidor.sh 127.0.0.1

* 1. Execute o arquivo arquivo Cliente.rb, em Cliente >> CamadaAplicacao.

No terminal, digite:

ruby Cliente.rb 127.0.0.1 *<consultaDNS>*

Em *<consultaDNS>* deve ser inserido o domínio ou o IP a ser consultado.

1. **Camada de Transporte**
   1. Instalar C#.
   2. Execute o arquivo arquivo Program.cs, em Servidor >> CamadaTransporte.
   3. Execute o arquivo arquivo Program.cs, em Cliente >> CamadaTransporte.
2. **Camada de Rede**
3. **Camada Física**
   1. Execute o arquivo arquivo Servidor.php, em Servidor >> CamadaFisica , com o comando "php Servidor.php"
   2. Execute o arquivo arquivo Cliente.php, em Cliente >> CamadaFisica , com o comando "php Cliente.php"
   3. Os resultados da execução estão descritos no arquivo "log.txt"

# **Resultados**

Um exemplo de resultado, encontra-se a seguir. Tal resultado foi obtido através dos registros no arquivo log.txt.

10 September 2018 20:39:32 [Física: Servidor] Socket com a camada física criado.

10 September 2018 20:39:32 [Física: Servidor] Vinculando um nome para o socket.

10 September 2018 20:39:32 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:35 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:35 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:35 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:35 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:35 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:35 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:35 [Física: Servidor] Enviando limite máximo.

10 September 2018 20:39:35 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:35 [Física: Cliente] Resposta recebida.

10 September 2018 20:39:35 [Física: Cliente] Colisão! Tentativa 1.

10 September 2018 20:39:38 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:38 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:38 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:38 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:38 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:38 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:38 [Física: Servidor] Mensagem {www.google.com} recebida.

10 September 2018 20:39:38 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:38 [Física: Cliente] Resposta recebida.

10 September 2018 20:39:44 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:44 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:44 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:44 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:44 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:44 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:44 [Física: Servidor] Enviando limite máximo.

10 September 2018 20:39:44 [Física: Cliente] Resposta recebida.

10 September 2018 20:39:44 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:44 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:44 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:44 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:44 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:44 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:44 [Física: Servidor] Mensagem {www.google.com} recebida.

10 September 2018 20:39:44 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:44 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:44 [Física: Cliente] Resposta recebida.

10 September 2018 20:39:47 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:47 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:47 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:47 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:47 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:47 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:47 [Física: Servidor] Enviando limite máximo.

10 September 2018 20:39:47 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:47 [Física: Cliente] Resposta recebida.

10 September 2018 20:39:47 [Física: Cliente] Colisão! Tentativa 1.

10 September 2018 20:39:50 [Física: Cliente] Colisão! Tentativa 2.

10 September 2018 20:39:55 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:55 [Física: Cliente] Protocolo ARP o ip 127.0.0.1 pertence ao MAC d0:df:9a:c4:07:ab.

10 September 2018 20:39:55 [Física: Cliente] Socket com a camada física criado.

10 September 2018 20:39:55 [Física: Cliente] Mensagem enviada.

10 September 2018 20:39:55 [Física: Servidor] Conexão aceita.

10 September 2018 20:39:55 [Física: Servidor] Quadro recebido.

10 September 2018 20:39:55 [Física: Servidor] Mensagem {www.google.com} recebida.

10 September 2018 20:39:55 [Física: Servidor] Ouvindo a conexão.

10 September 2018 20:39:55 [Física: Cliente] Resposta recebida.

# **Referências Bibliográficas**

[1] SILVA, C. Servidor DNS: Veja como escolher o melhor para acelerar sua navegação. Disponível em: <<https://canaltech.com.br/internet/veja-como-escolher-o-melhor-servidor-dns-para-acelerar-sua-navegacao/>>. Acesso em: 29 set. 2018.

[2] SILVA, Adelson de Paula. Princípios de Comunicação de Dados - Arquitetura de Redes de Computadores. Centro Federal de Educação Tecnológica de Minas Gerais, Engenharia da Computação, 2018.

[3] KUROSE, James F.; ROSS, K. W.. Redes de computadores e a Internet: uma abordagem top-down. Tradução Daniel Vieira, 6. ed. – São Paulo: Pearson Education do Brasil, 2013.

[4] Ruby-Doc. Help and documentation for the Ruby programming language. Disponível em: <https://ruby-doc.org/stdlib-2.5.0/libdoc/socket/rdoc/Socket.html>.

[5] RubyDoc.info. YARD Documentation Server. Disponível em: <<https://www.rubydoc.info/stdlib/resolv/Resolv/DNS>>.

[6] IETF. Request for Comments: 1034 - DOMAIN NAMES - CONCEPTS AND FACILITIES. Network Working Group, nov. 1987. Disponível em: <<https://www.ietf.org/rfc/rfc1034.txt>>.

[7] IETF. Request for Comments: 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. Network Working Group, nov. 1987. Disponível em: <<https://www.ietf.org/rfc/rfc1035.txt>>.

[8] MICROSOFT. Network Programming in the .NET Framework. Disponível em: <<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/index>>.

[9] MICROSOFT. Guia de programação em C#. Disponível em: <<https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/>>.

[10] VIVA O LINUX. Configurando DOTNET Core e instalando VSCode no Linux. Disponível em:<<https://www.vivaolinux.com.br/artigo/Configurando-DOTNET-Core-e-instalando-VSCode-no-Linux>>

[11] MICROSOFT. .NET Core e .NET Framework. Disponível em: <https://www.microsoft.com/net/download>.

[12] VIVA O LINUX. Pré-requisitos para o .NET Core no Linux. Disponível em: <<https://docs.microsoft.com/pt-br/dotnet/core/linux-prerequisites?tabs=netcore2x>>

[13] MONO-PROJECT. Disponível em: < <https://www.mono-project.com/download/stable/#download-lin-debian> >

[14] RFC 793. Transmission Control Protocol - Protocol Specification. Setembro, 1981. Disponível em: <<https://tools.ietf.org/html/rfc793#page-15>>.

[15] RFC 768. User Datagram Protocol. Disponível em: <https://www.ietf.org/rfc/rfc768.txt>.

[16] RFC 791. Internet Protocol. Disponível em: https://tools.ietf.org/html/rfc791.

[17] RStudio. Disponível em: https://www.rstudio.com/>.

[18] R-Project. Disponível em: <https://www.r-project.org/other-docs.html>.

[19] R-Bloggers. Disponível em: <https://www.r-bloggers.com/using-r-to-communicate-via-a-socket-connection/ >.

[20] RFC 790. Assigned Numbers. Disponível em: <https://tools.ietf.org/html/rfc790>.

[21] FOROUZAN, B. A. Comunicação de dados e redes de computadores. Tradução de Ariovaldo Griesi. Quarta Edição, Mc Graw Hill, Bookman, 2008.

[22] Geeksforgeeks. Computer Network | Ethernet Frame Format. Disponivel em: <<https://www.geeksforgeeks.org/computer-network-ethernet-frame-format/>>.

[23] PHP Manual. Disponivel em: <<http://www.php.net/docs.php>>.