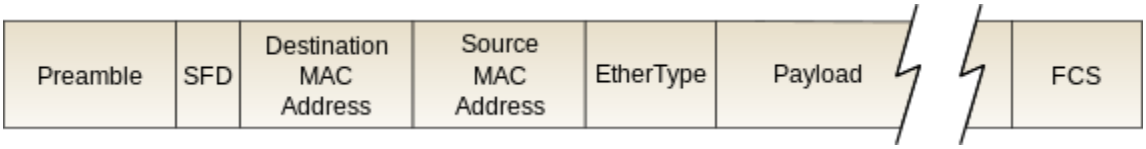
Redes I – TP - Implementação de pilha de protocolos DNS

Jonathan Henrique, Rúbio Torres e Igor Miranda

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.

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 sequencia 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 “10101010101010101010101010101010101” (o polinômio gerador deve ter 1 bit a mais que o resultado desejado)

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.

```
C:\Users\Jonathan\Documents\Trabalhos\Redes I\layers_of_a_network-\src\PhysicalLayer>python test.py
Physical Layer Test- Python
Server running

Commands:
test_decode: Receive an encoded test file and decode it
test_encode <IP>: Encode a test file and send it to IP
exit: Exit the program

Command >>
```

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.

```
2019-04-10 18:20:06 >> Server listening...
2019-04-10 18:23:34 >> Got connection from 127.0.0.1
2019-04-10 18:23:34 >> Server poke

2019-04-10 18:23:34 >> Server listening...
2019-04-10 18:23:42 >> Got connection from 127.0.0.1
2019-04-10 18:23:42 >> Arp: Searhing MAC for 192.168.15.7... 54271ef141db
2019-04-10 18:23:42 >>
Processed PDU
Message:
192.168.15.7::Arquivo enviado pela rede!

Result:
10101010101010101010101010101010101010101010101010101010101010101010101011010101000010011100011110111100010100000111011011111110
00000101000000110011010000111101001010101110000000000101000001100010011100100110010001011100011000100110110001110000010111000110
10111000110001001101010010111000110111001110100011101001000001011100100111000101110101011010010111011001101111001000000111001001
00110010101101110011101110011010010110000101100100011011110010000001110000011001010110110001100001001000000111001001100101010110
```

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

```
2019-04-10 18:25:22 >> Server listening...
2019-04-10 18:25:28 >> Got connection from 127.0.0.1
2019-04-10 18:25:28 >>
Read PDU
10101010101010101010101010101010101010101010101010101010101010101010101011010101000010011100011110111100010100000111011011111110
00000101000000110011010000111101001010101110000000000101000001100010011100100110010001011100011000100110110001110000010111000110
10111000110001001101010010111000110111001110100011101001000001011100100111000101110101011010010111011001101111001000000111001001
00110010101101110011101110011010010110000101100100011011110010000001110000011001010110110001100001001000000111001001100101010110

Result:
{
  "begin": "10101010101010101010101010101010101010101010101010101010101010101010101011",
  "destination": "acc66211e6a0",
  "origin": "f82819a1e957",
  "size": 320,
  "payload": "192.168.15.1::Arquivo enviado pela rede!",
  "crc": "11100011110111010011111101110111"
}

CRC check: Success!

2019-04-10 18:25:28 >> Successfully got the file
```

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

```python
def crc_remainder(input_bitstring):
    polynomial_bitstring = '10101010101010101010101010101010'
    len_input = len(input_bitstring)
    initial_padding = '0' * (len(polynomial_bitstring) - 1)
    input_padded_array = list(input_bitstring + initial_padding)
    while '1' in input_padded_array[:len_input]:
        cur_shift = input_padded_array.index('1')
        for i in range(len(polynomial_bitstring)):
            input_padded_array[cur_shift + i] = str(int(polynomial_bitstring[i] != input_padded_array[cur_shift + i]))
    return ''.join(input_padded_array)[len_input:]

def crc_check(input_bitstring, check_value):
    polynomial_bitstring = '10101010101010101010101010101010'
    len_input = len(input_bitstring)
    initial_padding = check_value
    input_padded_array = list(input_bitstring + initial_padding)
    while '1' in input_padded_array[:len_input]:
        cur_shift = input_padded_array.index('1')
        for i in range(len(polynomial_bitstring)):
            input_padded_array[cur_shift + i] = str(int(polynomial_bitstring[i] != input_padded_array[cur_shift + i]))
    return ('1' not in ''.join(input_padded_array)[len_input:])
```

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

```python
def get_destination_mac(ip):
    mac_destination = mac_table.get(ip)

    if not mac_destination:
        mac_destination = arp(ip)
        mac_table[ip] = mac_destination
    else:
        print(show_timestamp() + "Got MAC address from table: ", end='')
    print (mac_destination)

    return mac_destination

def arp(ip):
    print(show_timestamp() + "Arp: Searhing MAC for {}... ".format(ip), end='')
    result = subprocess.run(['arp', '-a', ip], stdout=subprocess.PIPE).stdout.decode('latin')
    pattern = re.compile(r'(?:[0-9a-fA-F]-?){12}')
    mac_list = re.findall(pattern, result)
    if len(mac_list) < 1:
        raise Exception('MAC not found :/')
    mac = mac_list[0].replace('-', '')
    return mac
```

Código do ARP

```python
def send_data(data, destination_ip):
    collision = random.randint(1, 100) <= 5
    while collision:
        print(show_timestamp() + "Collision! Waiting...")
        time.sleep(random.randint(1,100)/100)
        collision = random.randint(1, 100) <= 5
```

Código de colisão

```
def mount_frame(data):
    data = data.decode('latin')
    begin = '10101010101010101010101010101010101010101010101010101010101011'
    origin = hex2bin(host_mac)
    destination_ip = data.split('::')[0]
    payload = str2bin(data)
    bin_size = int2bin(len(data))
    crc = crc_remainder(payload)

    destination = get_destination_mac(destination_ip)
    destination_bin = hex2bin(destination)

    result = begin + destination_bin + origin + bin_size + payload + crc

    print (show_timestamp() + "\nProcessed PDU\nMessage:\n{}\n\nResult:\n{}\n".format(data,result))
    return result, destination_ip

def unmount_frame(data):
    data = data.decode('latin')
    size = int(data[160:176], 2) * 8
    frame = {
        'begin': data[0:64],
        'destination': bin2hex(data[64:112]),
        'origin': bin2hex(data[112:160]),
        'size': size,
        'payload': bin2str(data[176:(176+size)]),
        'crc': data[(176+size):]
    }

    print (show_timestamp() + "\nRead PDU\n{}\n\nResult:".format(data))
    print (json.dumps(frame, indent=2))
    print ("\nCRC check: ", end='')

    if crc_check(data[176:(176+size)], frame['crc']):
        print ('Success!\n')
    else:
        print ('Fail :/\n')

    return frame
```

Montagem e “desmontagem” de quadros

Considerações Finais

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.