Projeto 3 - Realizar a interpretação de cabeçalhos IP e reconstruir datagramas IP fragmentados.

Grupo:
Guilherme Nishi Kanashiro - 628298
Leonardo Utida Alcântara - 628182
Rodolfo Krambeck Asbahr - 628042
Tiago Bachiega de Almeida - 628247

Como executar o programa:

A implementação da interpretação e reconstrução dos datagramas IPs foram feitas utilizando python 3. Para rodar o código em um terminal é preciso executar o seguinte comando:

```
$ sudo python3 etapa3.py
```

Feito isso, o programa irá iniciar o envio de pacotes ping (ICMP echo request) com payload grande para o endereço definido na variável “dest_addr”. No nosso caso esse endereço é definido em:

```
dest_addr = '127.0.0.1'
```

Como esse endereço é o de loopback, é necessário ajustar o mtu dele, pois o tamanho padrão é de 65536 bytes, o que impossibilita qualquer forma de fragmentação automática. Para fazer esse ajuste, é necessário usar o seguinte comando:

```
$ ip link set lo mtu 1500
```

Como funciona o programa:

Tratamento de Mensagens Fragmentadas

O código prepara um pacote de mais de 20000 bytes para enviar para o endereço destino, que no caso é loopback, através da função send_ping(). O envio é repetido várias vezes durante a execução do código.

O mesmo código recebe pacotes como descrito no próprio código. Após um pacote ser recebido, será realizado um tratamento de desfragmentação de mensagem através da função defrag().

Primeiro são verificadas as flags do cabeçalho do pacote, existem duas flags no cabeçalho: uma que diz que a mensagem não precisa de fragmentação, *Don't Fragment*, e outra que diz que é mais fragmentos irão chegar, *More Fragments*. Se nenhuma flag estiver ativa, quer dizer que o pacote é o último pacote de uma mensagem fragmentada. As flags no cabeçalho IPV4 são definidas por 3 bits mostrados abaixo (campo "Flags").

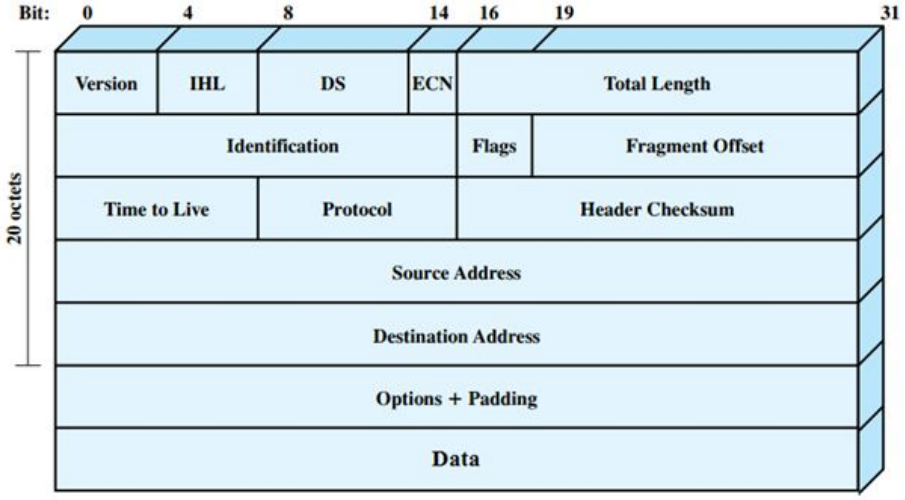

Figure IPv4 Header

Após a verificação do pacote fragmentado, ele vai ser adicionado, se ainda não foi, a um dicionário, chamado de "receivedPackages", que armazena pacotes de mensagens que ainda não foram desfragmentadas. O índice desse dicionário é uma tupla com o endereço de origem da mensagem, id da mensagem e offset da mensagem. Após essa adição ao dicionário, será verificado se todos os pacotes dessa mensagem já chegaram, através da função checkAllPackets(). Essa função percorrerá o dicionário inteiro verificando quantos pacotes são da mesma mensagem, verificando o endereço de origem e id da mensagem. Se for encontrado o último pacote da mensagem, é possível descobrir em quantos pacotes a mensagem se dividiu. Após a contagem de pacotes acabar, é verificado se é conhecida a quantidade total de pacotes e se essa quantidade é a mesma a de pacotes já recebidos.

Se todos os pacotes foram recebidos, o código irá montar a mensagem através da função appendPackets(). Essa função consiste no acesso a todos os pacotes da mensagem de forma sequencial. De início, o código irá acessar o pacote de offset 0 e o adicionar em uma estrutura vazia. Em seguida, serão acessados os pacotes seguintes aumentando o offset em

185 a cada iteração (os offsets sempre são múltiplos de 185) e concatenando o conteúdo do pacote à estrutura com os pacotes anteriores. Quando o último pacote for acessado, quer dizer que a montagem acabou e o código pode retornar a estrutura com o conteúdo completo da mensagem. Assim, a mensagem desfragmentada e completa é obtida e o programa está pronto para receber um novo pacote ou mensagem.

Timeout

Sempre que um pacote tem seu processamento finalizado na função *defrag*, um timeout é disparado para tratar o não envio dos pacotes seguintes. Este timeout é chamado com a função *asyncio.get_event_loop().call_later()*, que após expirar o tempo de espera para pacotes subsequentes chama a função *discard_packet()* que, a partir da identificação da mensagem, descarta no dicionário de pacotes recebidos todos os pacotes que foram recebidos até então correspondentes à mensagem deste fragmento que não veio. Em outras palavras, caso um novo fragmento ou mensagem não chegue a tempo, o programa desiste de tentar desfragmentar aquela mensagem e parte para a próxima.

Sempre que um pacote é recebido e ele entra na função *defrag()* e o timeout é resetado, pois um novo fragmento chegou.

```
def discard_packet(idTuple):

    global counter
    global testTimeout

    #reseta o counter
    if testTimeout:
        counter = 0

    print("DESCARTANDO ", idTuple, " POR TIMEOUT\n")

    #copia o dicionario para iterar
    iterDict = receivedPackages.copy()
    for tuple in iterDict:
       if idTuple[0] == tuple[0] and idTuple[1] == tuple[1] and
            idTuple[2] == tuple[2] and idTuple[3] == tuple[3] and idTuple[4] == tuple[4]:
           del receivedPackages[tuple]
```

Teste de Desfragmentação:

```
[root@localhost Projeto3]# python3 etapa3.py
enviando ping
recebido pacote de 1500 bytes
Recebidos 1 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 2 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 3 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 4 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 5 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 6 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 7 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 8 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 9 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 10 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 11 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 12 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 13 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
recebido pacote de 784 bytes
Recebidos todos os 14 pacotes da mensagem 45402 vinda de 127 . 0 . 0 . 1
Mensagem desfragmentada!
```

A função *send_ping()* envia uma mensagem com 20.004 bytes, como o tamanho máximo dos pacotes IP são de 1.500 bytes a mensagem é fragmentada é enviada em pacotes de até 1.500 bytes. Na figura acima, a mensagem 45402 foi fragmentada em 14 pacotes. E após o recebimento de cada pacote, é feita a desfragmentação e a mensagem original é remontada.

Teste de Timeout:

Caso o tempo de espera por um pacote seja atingido, ou seja, ocorre o timeout, a mensagem é descartada. Como é mostrado na figura abaixo, onde o terceiro pacote não é recebido dentro do tempo. Para o teste, foi feito um contador que itera a cada pacote recebido. Quando o contador fica maior do que 3 o programa retorna e não obtém o pacote até que o timeout termine, simulando a perda de um fragmento. Assim, o programa deleta todos os demais fragmentos da mensagem que não foi recebida em tempo limite.

```
[root@localhost Projeto3]# python3 etapa3.py
enviando ping
recebido pacote de 1500 bytes
Recebidos 1 pacotes da mensagem 54866 vinda de 127 . 0 . 0 . 1
recebido pacote de 1500 bytes
Recebidos 2 pacotes da mensagem 54866 vinda de 127 . 0 . 0 . 1
*****************DESCARTANDO  (127, 0, 0, 1, 54866, 185)  POR TIMEOUT*****************
```

```python
def raw_recv(recv_fd):

    #Faz o teste do timeout
    #A ideia e que quando o contador fique com o valor
    #maior que 3 ele fique preso aqui por tempo o
    #suficiente para o discard executar o discard_packet
    if testTimeout:
        global counter
        global exitTest
        counter = counter + 1
        #no 3 pacote ele vai simular uma perda
        if counter > 3:
            #return ate chamar discar_packet
            return

    packet = recv_fd.recv(1500)
    print('recebido pacote de %d bytes' % len(packet))
    defragMSG = defrag(packet)
```