



## Relatório de Aula Prática – Redes de Computadores

Título: UDP

Aluno: Vitor Mayorca Camargo

Data: 11/09/2024

---

### 1. INTRODUÇÃO

A internet surgiu na década de 1960 como ARPANET, inicialmente usada para comunicação militar e acadêmica nos EUA (LEINER, *et al.* 2009). Em 1989, a criação da World Wide Web democratizou o acesso e compartilhamento de informações por meio de navegadores web, permitindo que pessoas comuns usassem a internet de forma fácil e rápida (MONTEIRO, 2001). As informações transmitidas na internet seguem protocolos rigorosos, como o TCP, o IP, e o UDP – surgidos da necessidade de estabelecer regras e padrões para unificar as diferentes redes e máquinas que formam a internet (CORONA, 2004).

Kurose e Ross (2021) os hardwares, softwares, e protocolos de rede são divididos em “camadas”: Aplicação, Transporte, Rede, Enlace, e Física. O protocolo IP, previamente mencionado, normalmente está relacionado à camada de rede. Já a camada de transporte possui dois principais protocolos: o TCP e o UDP.

O protocolo TCP (Transmission Control Protocol) foi projetado para fornecer uma comunicação confiável entre dois computadores, reduzindo ou aumentando a taxa de transmissão de dados de forma dinâmica, com o fim de evitar perda de dados. (CAMPISTA, *et al.* 2010). Já o protocolo IP (Internet Protocol) define a base para a transação, tráfego e reconhecimento de dados em uma rede, definindo um "Endereço IP", que é um número único dado a cada máquina ou "host" na rede (CORONA, *et al.* 2004).

O protocolo UDP (User Datagram Protocol) é descrito na RFC 768, e possui como principal característica o envio de pacotes sem estabelecer uma conexão prévia, sendo muito rápido, mas sem garantia de que os pacotes chegarão em ordem (TANENBAUM, 2003). Os pacotes UDP possuem um cabeçalho de 4 campos de 2 bytes: Source port, que indica a porta de origem do pacote; Destination port, que armazena a porta do destinatário, e indica a aplicação que receberá o pacote; UDP Length, que indica o tamanho total do pacote UDP; e o UDP Checksum, que serve como medida de segurança para o destinatário identificar se o pacote chegou com erros (KUROSE, ROSS. 2021).

Por não ser orientado à conexão, o protocolo UDP acaba sendo mais ágil, e é comumente utilizado em aplicações em tempo real. Porém, seu único mecanismo de controle de erros é o descarte de pacotes corrompidos (DOURADO, COSTA. 2008). Ele usa o protocolo IP para transportar mensagens. Porém, para distinguir entre diferentes destinos dentro de um host, ele usa o mecanismo de “portas”. Elas funcionam como uma interface, um multiplexador que gerencia o tráfego das informações do protocolo IP (DINIZ, JUNIOR. 2014).

Além disso, o protocolo UDP é muito utilizado para a realização de multicasts, ou seja, quando se quer enviar um mesmo pacote para vários destinos diferentes. Hosts que querem receber um multicast particular precisam se registrar em um grupo usando o protocolo IGMP (Internet Group Management Protocol). Multicasts não são enviados para redes sem nenhum host neste grupo (NAGEL, 2004).

Servidores UDP simples podem ser implementados utilizando a linguagem de programação Python. Segundo Neto (2007), Python é uma linguagem interessante para essa função por ser uma linguagem de altíssimo nível, com tipagem dinâmica, sintaxe simples, e inúmeras bibliotecas que facilitam o processo de desenvolvimento de diferentes sistemas.

Os pacotes UDP gerados por servidores UDP são normalmente invisíveis ao usuário. Porém, eles podem ser visualizados usando ferramentas conhecidas como sniffers de rede. Essas ferramentas capturam e analisam o tráfego de rede, permitindo a visualização dos pacotes em tempo real. As duas ferramentas de sniffing de redes mais populares são o Wireshark e o Tcpdump (GOYAL, GOYAL. 2017). Além disso, a ferramenta *nslookup* também pode ser utilizada para requisitar e verificar todas as informações dos pacotes DNS (KUROSE, ROSS. 2021).

## 2. OBJETIVOS

A atividade prática realizada no dia 19/09/2024 teve como objetivo entender o funcionamento do protocolo UDP na prática. Isso foi feito por meio da implementação de um servidor e um cliente que utilizam esse protocolo, com o uso da linguagem de alto nível *Python*. O sniffer de rede *wireshark* também foi utilizado para auxiliar na visualização dos pacotes comutados na rede.

## 3. MATERIAL UTILIZADO

Os testes foram realizados numa máquina virtual, executando num notebook da marca DELL, modelo Vostro 3520, com um CPU Intel Core i7-1255U 1.70 GHz, 16Gb de memória RAM DDR4, placa de vídeo integrada Intel Iris Xe Graphics, placa de video dedicada NVIDIA GeForce MX550, adaptador de rede modelo Intel(R) Wi-Fi 6 AX201 160MHz, unidade de disco SSD NVMe ADATA 512Gb, com o sistema operacional Windows 11 Home versão 23H2.

A máquina virtual foi criada com o software Oracle VM Virtualbox, versão 7.0.12. Dentro dela, foi executado o sistema operacional MX Linux versão 6.1.0-21-amd64, uma distro baseada em Debian. O ambiente virtual foi conectado à placa de rede do notebook por meio do modo Bridge. Mais especificações da máquina virtual estão explícitas na figura 1.

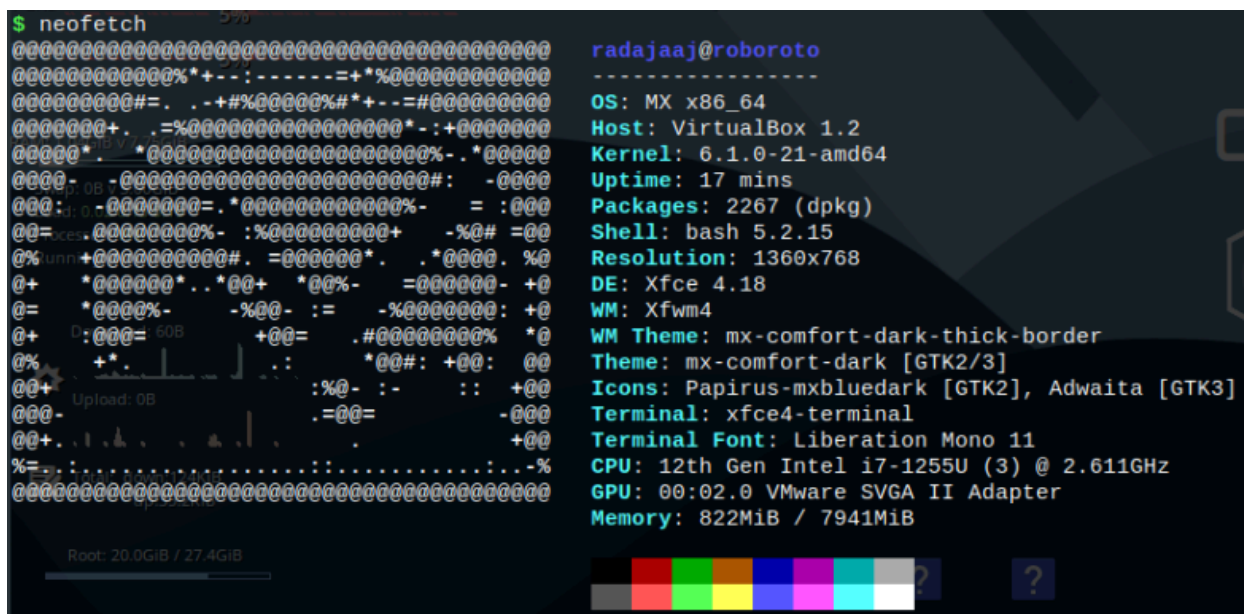


Figura 1: Ambiente virtual após a execução do comando *neofetch*.

No ambiente virtual Linux, foram instalados os pacotes *wireshark* e *python-is-python3*. Os testes foram feitos na UNIOESTE, durante a aula prática do dia 19/09/2024.

## 4. METODOLOGIA

### 4.1 Implementação do Servidor UDP:

O servidor UDP foi implementado conforme o código fonte disponibilizado pelo professor durante a aula prática. Ele pode ser visualizado na Figura 2.

```
python - Servidor UDP

1  import socket
2
3  # Criação do socket UDP
4  udp_server_socket = socket.socket(socket.AF_INET,
5                                   socket.SOCK_DGRAM)
6
7  # Ligação do socket ao endereço e porta
8  # IPv4='localhost' ou '127.0.0.1', IPv6='::1'
9  server_address = ('localhost', 12345)
10
11 udp_server_socket.bind(server_address)
12
13 print("Servidor UDP aguardando mensagens...")
14
15 # Recebe dados do cliente
16 while True:
17     # Buffer de 1024 bytes
18     data, address = udp_server_socket.recvfrom(1024)
19     print(f"Mensagem recebida: {data.decode()} de {address}")
20
21     # Enviar uma resposta opcional para o cliente
22     response = "Mensagem recebida com sucesso"
23     udp_server_socket.sendto(response.encode(), address)
24     print(f"Resposta enviada para {address}")
```

Figura 2: Código fonte em python do servidor UDP.

### 4.2 Implementação do Cliente UDP:

De forma similar ao servidor, o cliente também foi disponibilizado pelo professor durante a aula prática, e pode ser checado na Figura 3.

#### python - Cliente UDP

```
1 import socket
2
3 # Criação do socket UDP
4 udp_client_socket = socket.socket(socket.AF_INET,
5                                   socket.SOCK_DGRAM)
6
7 # Endereço do servidor e porta
8 # No mesmo host, IPv4='localhost' ou '127.0.0.1', IPv6=':::1'
9 dest_ip = '127.0.0.1' # Altere para o IP de destino
10 dest_port = 12345 # Porta de destino
11
12 server_address = (dest_ip, dest_port)
13
14 # Mensagem a ser enviada ao servidor
15 message = "Ola, servidor UDP!"
16 udp_client_socket.sendto(message.encode(), server_address)
17
18 # Recebe resposta do servidor
19 data, server = udp_client_socket.recvfrom(1024)
20 print(f"Resposta do servidor: {data.decode()}")
21
22 # Fechando o socket do cliente
23 udp_client_socket.close()
```

**Figura 3:** Código fonte em python do cliente UDP.

### 4.3 Testando o Servidor:

Com os códigos implementados, iniciaram-se os testes dos mesmos. Primeiro, foi iniciada uma captura com o sniffer de rede *wireshark*. Depois, o servidor e o cliente foram executados no ambiente do Linux. No *wireshark*, os pacotes foram filtrados por “udp && udp.port==12345”, e o primeiro pacote capturado foi escolhido e analisado.

### 4.4 UDP com Raw Sockets:

Agora, o código fonte do cliente é alterado para se obter um controle total sobre o conteúdo dos cabeçalhos dos pacotes UDP enviados. Novamente, o código fonte foi disponibilizado pelo professor, e pode ser visualizado na Figura 4. O novo cliente foi então testado, e os pacotes comutados foram capturados e analisados no *wireshark*.

## python - Cliente UDP raw socket

```
1
2 import socket
3 import struct
4
5 def checksum(msg):
6     s = 0
7     # Somar as palavras de 16 bits
8     for i in range(0, len(msg), 2):
9         w = (msg[i] << 8) + (msg[i+1])
10        s = s + w
11    s = (s >> 16) + (s & 0xffff)
12    s = s + (s >> 16)
13    return ~s & 0xffff
14
15 # Endereço IP e porta do destino
16 dest_ip = '127.0.0.1' # Altere para o IP de destino
17 source_ip = '127.0.0.1' # IP de origem
18
19 # Criar socket UDP usando IPPROTO_UDP (apenas cabeçalho UDP)
20 raw_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW,
21                             socket.IPPROTO_UDP)
22
23 # Construção do cabeçalho UDP
24 source_port = 1234 # Porta de origem
25 dest_port = 12345 # Porta de destino
26 data = b"Ola, servidor UDP!" # Dados a serem enviados
27 udp_len = 8 + len(data) # Comprimento do cabeçalho UDP + dados
28 udp_check = 0 # Inicialmente, checksum é 0
29
30 # Cabeçalho UDP sem checksum
31 udp_header = struct.pack('!HHHH', source_port, dest_port,
32                             udp_len, udp_check)
33
34 # Pseudo-cabeçalho para calcular o checksum UDP
35 pseudo_header = struct.pack('!4s4sBBH',
36                             socket.inet_aton(source_ip),
37                             socket.inet_aton(dest_ip), 0,
38                             socket.IPPROTO_UDP, udp_len)
39 pseudo_packet = pseudo_header + udp_header + data
40
41 # Calculando o checksum UDP
42 udp_check = checksum(pseudo_packet)
43
44 # Atualizando o cabeçalho UDP com o checksum correto
45 udp_header = struct.pack('!HHHH', source_port, dest_port,
46                             udp_len, udp_check)
47
48 # Pacote final: cabeçalho UDP + dados
49 packet = udp_header + data
50
51 # Enviando o pacote
52 raw_socket.sendto(packet, (dest_ip, dest_port))
53 print(f"Pacote UDP enviado para {dest_ip}")
54
55 # Fechando o socket do cliente
56 raw_socket.close()
```

**Figura 4:** Código fonte em python do cliente UDP Raw Socket.

#### 4.5 UDP Multicast:

Por fim, foi feita mais uma alteração nos códigos fonte do client e do servidor, com o fim de permitir a transmissão e recebimento de pacotes por meio de multicast, enviando mensagens a um grupo de dispositivos. Agora, tanto o servidor quanto o cliente se inscrevem no grupo multicast. Os códigos fontes do cliente e do servidor estão disponíveis nas figuras 5 e 6, respectivamente.

##### python - Servidor UDP Multicast

```
1  import socket
2  import struct
3  import time
4
5  # Criação do socket UDP
6  # IPv4: 224.0.0.0 a 239.255.255.255, IPv6='ff02::1'
7  multicast_group = '224.1.1.1'
8  server_address = ('', 12345)
9
10 # Configuração do socket
11 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13 # Definindo TTL (Time to Live) para o pacote multicast
14 ttl = struct.pack('b', 1)
15 sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
16
17 # Enviando mensagens ao grupo multicast
18 try:
19     while True:
20         message = "Mensagem multicast UDP"
21         print(f"Enviando: {message}")
22         sent = sock.sendto(message.encode(),
23                             (multicast_group, 12345))
24         time.sleep(2) # Enviar a cada 2 segundos
25 finally:
26     sock.close()
```

Figura 5: Código fonte em python do servidor UDP Multicast.



### python - Cliente UDP Multicast

```
1 import socket
2 import struct
3
4 # Criação do socket UDP
5 multicast_group = '224.1.1.1' # Mesmo grupo do servidor
6 server_address = ('', 12345)
7
8 # Configuração do socket
9 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10
11 # Vincular o socket à porta do servidor
12 sock.bind(server_address)
13
14 # Informar que o cliente quer se juntar ao grupo multicast
15 group = socket.inet_aton(multicast_group)
16 mreq = struct.pack('4sL', group, socket.INADDR_ANY)
17 sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
18
19 # Receber mensagens do grupo multicast
20 try:
21     while True:
22         print("Aguardando mensagem multicast...")
23         data, address = sock.recvfrom(1024)
24         print(f"Recebido: {data.decode()} de {address}")
25 finally:
26     sock.close()
```

**Figura 6:** Código fonte em python do cliente UDP Multicast.

Depois disso, ambos os códigos foram executados e testados durante a atividade prática. Os pacotes comutados foram capturados e analisados com o uso do *wireshark*.

## 5. RESULTADOS E DISCUSSÃO

### 5.1 Testando o Servidor e o Cliente Básico:

Durante a prática, tanto o servidor quanto o cliente funcionaram corretamente, e foi possível se comunicar com outras máquinas dentro da rede. Porém, não foram tiradas prints do processo. Logo, o processo foi feito novamente na rede residencial do autor, com o envio de pacotes para o localhost. Uma print do estado do terminal após a execução do cliente e do servidor pode ser vista na figura 7, enquanto a captura do wireshark está na figura 8.

```
radajaa@roboroto:~/Redes/Pratica_4
$ python servidor.py
Servidor UDP ativo e aberto para mensagens!
('127.0.0.1', 42056) disse: Esta é uma mensagem!
Resposta enviada para ('127.0.0.1', 42056)

radajaa@roboroto:~/Redes/Pratica_4
$ python cliente.py
Resposta do server: Mensagem recebida!
$
```

Figura 7: Resultado da execução do servidor e do cliente básicos.

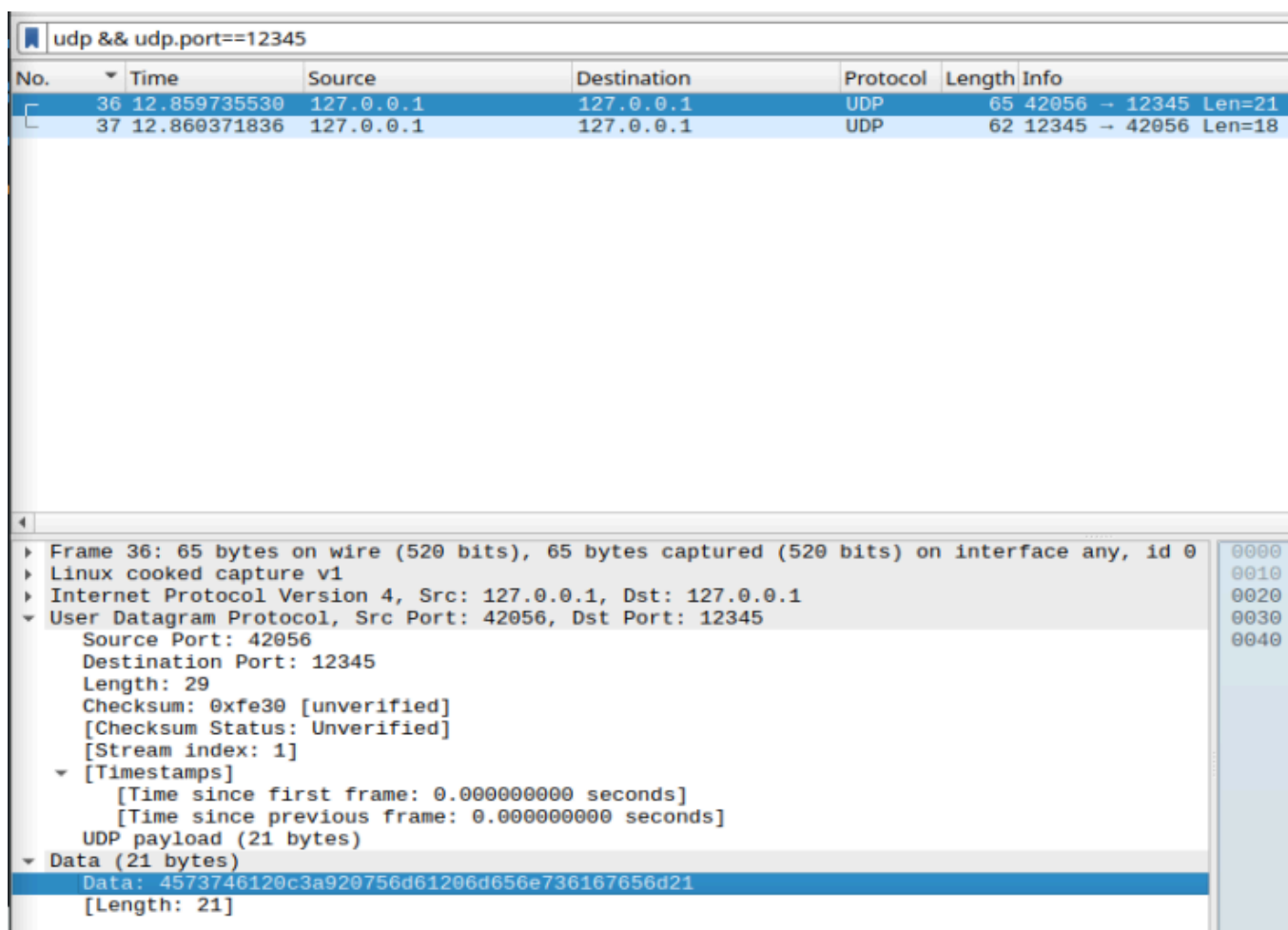


Figura 8: Pacotes capturados pelo sniffer durante a execução dos códigos básicos.

Ao selecionarmos o primeiro pacote, vemos a mensagem que o cliente enviou ao servidor. Temos todos os elementos do cabeçalho UDP: A porta de origem e destino, o checksum, e o tamanho dos dados.. O cabeçalho é seguido pela seção “Data”, com uma string hexadecimal que, quando traduzida para ASCII, se torna: “Esta é uma mensagem!”



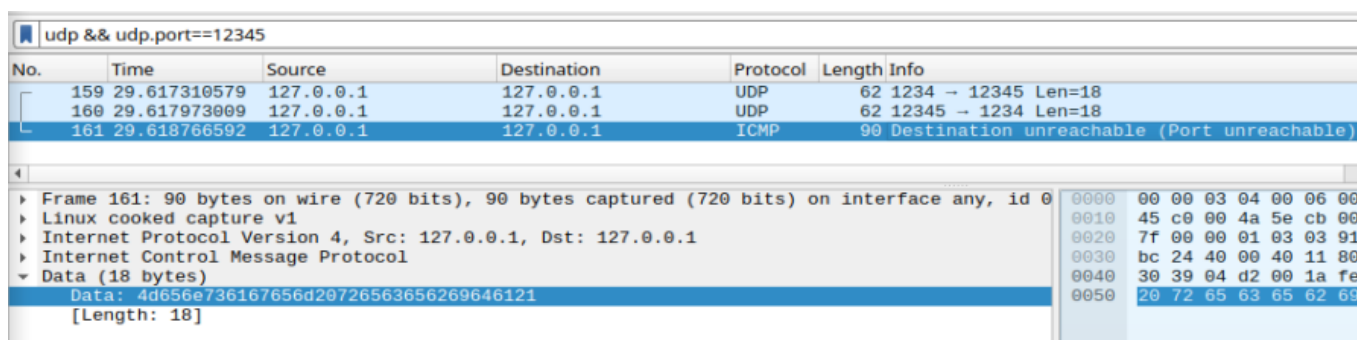
## 5.2 UDP com Raw Sockets:

No novo cliente, quase todo o pacote UDP é criado de forma mais manual. A Figura 9 (uma print do terminal) mostra que o processo ocorreu normalmente e sem falhas. Já a Figura 10 mostra a captura feita pelo sniffer de rede, que nos mostra que ocorreu um pequeno erro no processo.

```
radajaa@roboroto:~/Redes/Pratica_4
$ python servidor.py
Servidor UDP ativo e aberto para mensagens!
('127.0.0.1', 1234) disse: Ola, servidor UDP!
Resposta enviada para ('127.0.0.1', 1234)

radajaa@roboroto:~/Redes/Pratica_4
$ sudo python client_raw.py
[sudo] password for radajaa:
Pacote UDP enviado para 127.0.0.1
radajaa@roboroto:~/Redes/Pratica_4
$
```

Figura 9: Resultado da execução do cliente com raw sockets.



No.	Time	Source	Destination	Protocol	Length	Info
159	29.617310579	127.0.0.1	127.0.0.1	UDP	62	1234 → 12345 Len=18
160	29.617973009	127.0.0.1	127.0.0.1	UDP	62	12345 → 1234 Len=18
161	29.618766592	127.0.0.1	127.0.0.1	ICMP	90	Destination unreachable (Port unreachable)

Frame 161: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface any, id 0	
Linux cooked capture v1	
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	
Internet Control Message Protocol	
Data (18 bytes)	
Data: 4d656e736167656d20726563656269646121	0000 00 00 03 04 00 06 00
[Length: 18]	0010 45 c0 00 4a 5e cb 00
	0020 7f 00 00 01 03 03 91
	0030 bc 24 40 00 40 11 80
	0040 30 39 04 d2 00 1a fe
	0050 20 72 65 63 65 62 69

Figura 10: Pacotes capturados pelo sniffer durante a execução do cliente com raw sockets.

Percebemos que, desta vez, a resposta do servidor não foi capaz de chegar ao seu destino. Após uma análise rápida do código, vemos que o cliente com raw sockets não possui a função “udp\_client\_socket.recvfrom()”, o que faz com que ele não seja capaz de receber as respostas de confirmação do servidor.

## 5.3 UDP Multicast:

Por fim, tanto o servidor quanto o cliente multicast foram implementados e executados. Apesar de o servidor enviar corretamente os pacotes, percebeu-se que o cliente não os estava recebendo. Após uma rápida pesquisa na internet, descobriu-se que o culpado era o firewall do Linux, que foi temporariamente desativado com o comando “sudo ufw disable”. Depois disso, os pacotes foram recebidos normalmente, conforme visto na Figura 11. A captura do sniffer de rede está disponível na Figura 12.

```
radajaa@roboroto:~/Redes/Pratica_4
$ sudo python server_multicast.py
[sudo] password for radajaa:
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP
Enviando: Mensagem multicast UDP

radajaa@roboroto:~/Redes/Pratica_4
$ sudo python client_multicast.py
Aguardando mensagem multicast...
Recebido: Mensagem multicast UDP de ('10.0.2.15', 4879
5)
Aguardando mensagem multicast...
Recebido: Mensagem multicast UDP de ('10.0.2.15', 4879
5)
Aguardando mensagem multicast...
Recebido: Mensagem multicast UDP de ('10.0.2.15', 4879
5)
```

Figura 9: Resultado da execução do multicast.

udp && udp.port==12345							
No.	Time	Source	Destination	Protocol	Length	Info	
1	0.000000000	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
2	2.003131941	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
5	4.004928970	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
8	6.006724066	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
11	8.011116926	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
14	10.015321608	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
15	12.017282643	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
16	14.018530164	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
17	16.021369220	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22
18	18.023124966	10.0.2.15	224.1.1.1	UDP	66	48795 → 12345	Len=22

<ul style="list-style-type: none"> <li>Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface any, id 0</li> <li>Linux cooked capture v1</li> <li>Internet Protocol Version 4, Src: 10.0.2.15, Dst: 224.1.1.1</li> <li>User Datagram Protocol, Src Port: 48795, Dst Port: 12345 <ul style="list-style-type: none"> <li>Source Port: 48795</li> <li>Destination Port: 12345</li> <li>Length: 30</li> <li>Checksum: 0xed40 [unverified]</li> <li>[Checksum Status: Unverified]</li> <li>[Stream index: 0]</li> <li>[Timestamps] <ul style="list-style-type: none"> <li>[Time since first frame: 0.000000000 seconds]</li> <li>[Time since previous frame: 0.000000000 seconds]</li> </ul> </li> <li>UDP payload (22 bytes)</li> <li>Data (22 bytes) <ul style="list-style-type: none"> <li>Data: 4d656e7361676564206d756c74696361737420554450</li> <li>[Length: 22]</li> </ul> </li> </ul> </li> </ul>	0000 0010 0020 0030 0040
---	--------------------------------------

**Figura 11:** Pacotes capturados pelo sniffer durante a execução do multicast.

Durante a aula prática, também foi feito com sucesso um multicast entre as máquinas do laboratório. Porém, não foram tiradas fotos para comprovar tal evento.

## 6. CONCLUSÕES

A aula prática teve como objetivo clarificar alguns dos conceitos básicos relacionados ao funcionamento do protocolo UDP, bem como ter uma melhor visão do conteúdo interno dos pacotes enviados e recebidos por uma máquina que segue esse protocolo. Nesse sentido, conclui-se que o trabalho foi um sucesso, pois foi possível completar todas as atividades propostas, bem como obter um conhecimento básico sobre o funcionamento de servidores, clientes, multicasts, e estrutura de pacotes UDP.

O código implementado em python permitiu uma melhor visualização do funcionamento dos sockets usados na transferência de pacotes UDP, e também permitiu um melhor entendimento do processo de criação dos cabeçalhos e codificação dos dados enviados. Por fim, vale citar que a prática deixou clara a simplicidade do protocolo, o que o torna ideal para trocas de informação rápidas e diretas.

## BIBLIOGRAFIA

CAMPISTA, Miguel Elias M. et al. Interconexão de Redes na Internet do Futuro: Desafios e Soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 2010, p. 47-101, 2010.

CORONA, Adrián Estrada. *et al.* **Protocolos TCP/IP de internet**. 2004.

DOURADO, Ícaro Cavalcante; COSTA, Daniel G. **Utilizando UDP-Lite em Comunicações Multimídia em Tempo Real**. 2008

DINIZ, Pedro Henrique; JUNIOR, Nilton Alves. Ferramenta IPERF: geração e medição de Tráfego TCP e UDP. **Notas Técnicas**, v. 4, n. 2, 2014.

GOYAL, Piyush; GOYAL, Anurag. Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark. In: **2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)**. IEEE, 2017. p. 77-81.

KUROSE, James F.; ROSS, Keith W. **Redes de Computadores e a Internet: Uma abordagem top-down**. Trad. 8 ed. São Paulo: Francisco Araújo da Costa, 2021.

LEINER, Barry M. et al. **A brief history of the Internet**. ACM SIGCOMM computer communication review, v. 39, n. 5, p. 22-31, 2009.

MONTEIRO, Luís. **A internet como meio de comunicação: possibilidades e limitações**. In: Congresso Brasileiro de Comunicação. sn, 2001.

NAGEL, Christian et al. Pro. NET 1.1 **Network Programming**. Apress, 2004.

NETO, Jahyr Gonçalves. **Desenvolvimento de uma Plataforma Multimídia Utilizando a Linguagem Python**. 2007. Tese de Doutorado. Universidade Estadual de Campinas.

TANENBAUM, Andrew S. **Redes de Computadores**, 7ª Edição, Editora Campus, Rio de Janeiro – RJ, 2003.