

RELATÓRIO TRABALHO PRÁTICO 1

Thiago Andrade Ramalho

Redes de Computadores

1. INTRODUÇÃO

Neste relatório, serão apresentados exemplos práticos de comunicação de rede utilizando sockets, tanto para os protocolos UDP quanto TCP, conforme ilustrados no livro Redes de Computadores e a Internet (8ª edição) de James Kurose e Keith Ross. Sockets são uma ferramenta essencial para a comunicação em redes de computadores, funcionando como a interface entre a aplicação e a rede. Eles permitem que dois processos, em máquinas diferentes, troquem informações de maneira eficiente e controlada.

2. Execução dos códigos cliente e servidor e captura das telas de saída

1. UDP- Client

```
# O módulo socket forma a base de todas as comunicações de rede em
Python. Incluindo

# Nesta linha, podemos criar sockets dentro do nosso programa.

from socket import *

# Aqui, oferecemos uma cadeia contendo ou o endereço IP do servidor
(p. ex., "128.138.32.126") ou o nome

# de hospedeiro do servidor (p. ex., "cis.poly.edu"). Se usamos o
nome do hospedeiro, então

# uma pesquisa DNS será automaticamente realizada para obter o
endereço IP.
```

```
serverName = 'localhost'

# Define a variável inteira serverPort como 12000.

serverPort = 12000

# Esta linha cria o socket do cliente, denominado clientSocket. O
primeiro parâmetro

# indica a família do endereço; AF_INET indica que a rede subjacente
está usando IPv4.

# O segundo parâmetro indica que o socket é do tipo SOCK_DGRAM, o
que significa que é um socket UDP.

clientSocket = socket(AF_INET, SOCK_DGRAM )

# Quando esse comando input() é executado, o usuário no cliente
recebe o texto "Input lowercase sentence:".

# Então, o usuário usa seu teclado para digitar uma linha, que é
colocada na variável message

message = input('Input lowercase sentence :')

# Nesta linha, usamos o método encode() para convertamos a mensagem
do tipo cadeia para o tipo byte, para enviar bytes a um socket;

# sendto() acrescenta o endereço de destino (serverName, serverPort)
à mensagem

# e envia o pacote resultante pelo socket do processo, clientSocket.

clientSocket.sendto(message.encode(), (serverName, serverPort))
```

```
# Quando um pacote chega da Internet no socket do cliente, os dados são  
# colocados na variável modifiedMessage, e o endereço de origem do  
# pacote é colocado na variável serverAddress  
  
# A variável serverAddress tem tanto o endereço IP do servidor quanto  
# o número de porta do servidor.  
  
# O método recvfrom também toma o tamanho do buffer, 2048, como  
# entrada.  
  
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)  
  
# Aqui, usamos o método decode() para converter os bytes do pacote  
# de volta para uma cadeia,  
  
print(modifiedMessage.decode())  
  
# Ao final, fechamos o socket do cliente para encerrar a conexão, e  
# então o processo é concluído  
  
clientSocket.close()
```

2. UDP- Server

```
# O módulo socket forma a base de todas as comunicações de rede em  
# Python. Incluindo  
  
# Nesta linha, podemos criar sockets dentro do nosso programa.  
  
from socket import *  
  
# Define a variável inteira serverPort como 12000.
```

```
serverPort = 12000

# Esta linha cria o socket do server, denominado serverSocket. O
primeiro parâmetro

# indica a família do endereço; AF_INET indica que a rede subjacente
está usando IPv4.

# O segundo parâmetro indica que o socket é do tipo SOCK_DGRAM, o
que significa que é um socket UDP.

serverSocket = socket(AF_INET, SOCK_DGRAM )

# Esta linha vincula o número de porta 12000 ao socket do servidor.
Assim,

# em UDPServer, o código está designando um número de porta ao
socket.

# Com isso, quando alguém enviar um pacote à porta 12000 no endereço
IP do servidor, ele será direcionado a este socket

serverSocket.bind('', serverPort))

# UDPServer, então, entra em um

# laço while; o laço while permitirá que UDPServer receba e processe
pacotes dos clientes indefinidamente.

# No laço while, UDPServer espera um pacote chegar.

print("The server is ready to receive ")

while True:

    # Quando um pacote chega no
```

```

    # socket do servidor, os dados são colocados na variável
message, e o endereço de origem

    # é colocado na variável clientAddress. A variável client
Address contém o endereço IP

    # e o número de porta do cliente.

message , clientAddress = serverSocket.recvfrom(2048)

    # Esta linha é o núcleo da nossa aplicação simples. Ela apanha a
linha enviada pelo cliente

    # e, após converter a mensagem em uma cadeia, usa o método upper
() para passá-la para

    # letras maiúsculas

modifiedMessage = message.decode().upper()

    # Por fim, anexa o endereço do cliente (endereço IP e número de
porta) à mensagem

    # em letras maiúsculas (após converter a cadeia em bytes),
enviando o pacote resultante ao

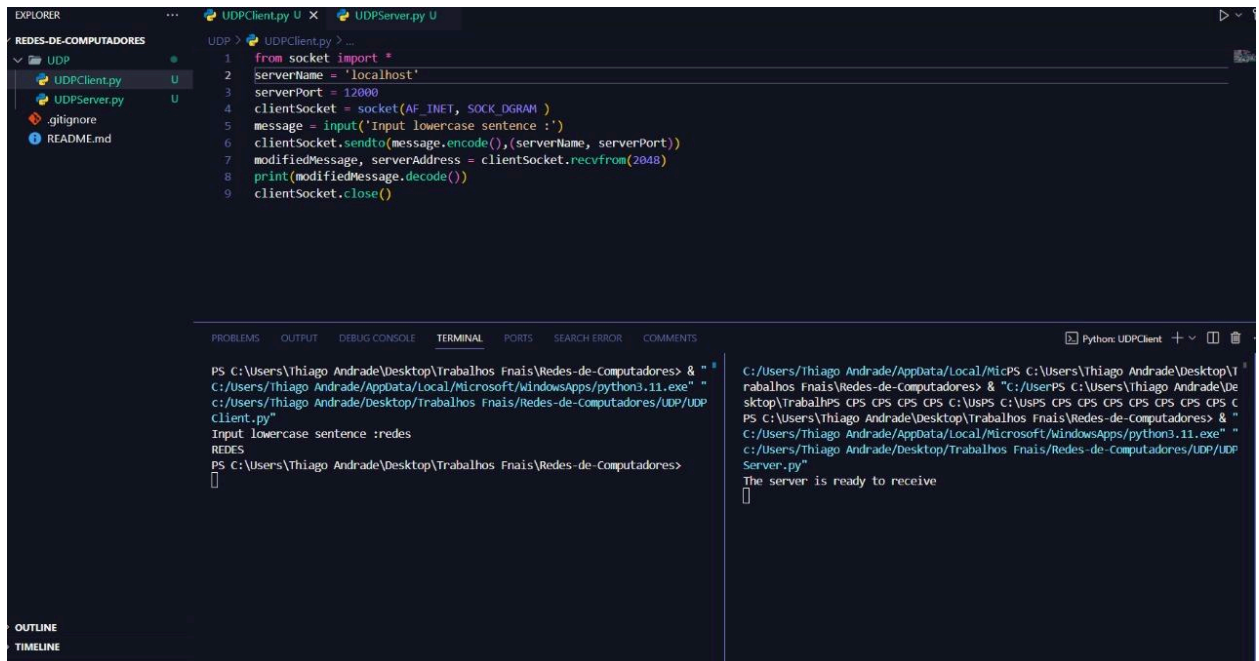
    # socket do servidor. A Internet, então, entregará o pacote ao
endereço do cliente.

    # Depois que o servidor envia o pacote, ele permanece no laço
while, esperando até que outro pacote UDP chegue

serverSocket.sendto(modifiedMessage.encode(),clientAddress)

```

3. UDP - Server Rodando



```
1 from socket import *
2 serverName = 'localhost'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_DGRAM)
5 message = input('Input lowercase sentence :')
6 clientSocket.sendto(message.encode(), (serverName, serverPort))
7 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
8 print(modifiedMessage.decode())
9 clientSocket.close()
```

```
PS C:\Users\Thiago Andrade\Desktop\Trabalhos Fnaís\Redes-de-Computadores> & "C:/Users/thiago Andrade/AppData/Local/Microsoft/windowsApps/python3.11.exe" "c:/Users/thiago Andrade/desktop/Trabalhos Fnaís/Redes-de-Computadores/UDP/UDP client.py"
Input lowercase sentence :redes
REDES
PS C:\Users\Thiago Andrade\Desktop\Trabalhos Fnaís\Redes-de-Computadores>

C:/Users/Thiago Andrade/AppData/Local/MicPS C:\Users\Thiago Andrade\Desktop\Trabalhos Fnaís\Redes-de-Computadores> & "C:/Users/thiago Andrade/AppData/Local/Microsoft/windowsApps/python3.11.exe" "c:/Users/thiago Andrade/desktop/Trabalhos Fnaís/Redes-de-Computadores/UDP/UDPServer.py"
The server is ready to receive
```

4. TCP - Client

```
# O módulo socket forma a base de todas as comunicações de rede em
Python. Incluindo

# Nesta linha, podemos criar sockets dentro do nosso programa.

from socket import *

# Aqui, oferecemos uma cadeia contendo ou o endereço IP do servidor
(p. ex., "128.138.32.126") ou o nome

# de hospedeiro do servidor (p. ex., "cis.poly.edu"). Se usarmos o
nome do hospedeiro, então

# uma pesquisa DNS será automaticamente realizada para obter o
endereço IP.

serverName = 'localhost'
```

```
# Define a variável inteira serverPort como 12000.

serverPort = 12000

# Essa linha cria o socket do cliente, denominado clientSocket. O
primeiro parâmetro

# indica que a rede subjacente está usando IPv4.

# o segundo parâmetro indica que o socket é do tipo SOCK_STREAM.

clientSocket = socket (AF_INET, SOCK_STREAM )

# Lembre-se de que, antes de um cliente poder enviar dados ao
servidor (e vice-versa) usando

# um socket TCP, primeiro deve ser estabelecida uma conexão TCP
entre eles, o que é feito

# por meio dessa linha. O parâmetro do método connect() é o endereço
do lado servidor da conexão

clientSocket.connect((serverName, serverPort))

# Essa linha obtém uma sentença do usuário. A cadeia sentence
continua a reunir caracteres

# até que o usuário termine a linha digitando um Enter

sentence = input('Input lowercase sentence : ')

# Essa linha envia a cadeia sentence pelo socket do cliente e para a
conexão TCP. E então deixa os bytes da cadeia

# sentence na conexão TCP. O cliente, então, espera para receber
bytes do servidor.
```

```

clientSocket.send(sentence.encode())

# Quando os caracteres chegam do servidor, eles são colocados na
cadeia modifiedSentence.

# Os caracteres continuam a ser acumulados em modifiedSentence até
que a linha termine com um caractere de Enter.

modifiedSentence = clientSocket.recv(1024)

# O método decode() é usado para converter os bytes do pacote de
volta para uma cadeia,

print('From Server: ', modifiedSentence.decode())

#Essa última linha fecha o socket e a conexão TCP entre cliente e
servidor.

clientSocket.close()

```

5. TCP - Server

```

# O módulo socket forma a base de todas as comunicações de rede em
Python. Incluindo

# Nesta linha, podemos criar sockets dentro do nosso programa.

from socket import *

# Define a variável inteira serverPort como 12000.

serverPort = 12000

```



```
# Essa linha cria o socket do server, denominado serverSocket. O
primeiro parâmetro

# indica que a rede subjacente está usando IPv4.

# o segundo parâmetro indica que o socket é do tipo SOCK_STREAM.

serverSocket = socket(AF_INET, SOCK_STREAM )


# O serverSocket será o socket de entrada. Depois de estabelecer
essa

# porta de entrada, ele vai ficar esperando e escutando até que
algum cliente bata à porta

serverSocket.bind('', serverPort))


# Essa linha faz com que o servidor escute as requisições de conexão
TCP do cliente.

# O parâmetro especifica o número máximo de conexões em fila (pelo
menos 1)

serverSocket.listen(1)


# Aqui, o programa entra em um loop infinito. Ele fica aguardando
conexões de clientes

print('The server is ready to receive')

while True :

    # Quando o cliente bate a essa porta, o programa chama o método
accept() para o serverSocket,
```

```
# que cria um novo socket no servidor, chamado connectionSocket,
dedicado a

# esse cliente específico. Cliente e servidor, então, completam
a apresentação, criando uma

# conexão TCP entre o clientSocket do cliente e o
connectionSocket do servidor.

# Após estabelecer a conexão TCP, cliente e servidor podem
enviar bytes um para o outro por

# ela. Com TCP, todos os bytes enviados de um lado têm garantias
não apenas de que chegarão

# ao outro lado, mas também na ordem

connectionSocket, addr = serverSocket.accept()

sentence = connectionSocket.recv(1024).decode()

capitalizedSentence = sentence.upper()

connectionSocket.send(capitalizedSentence.encode())

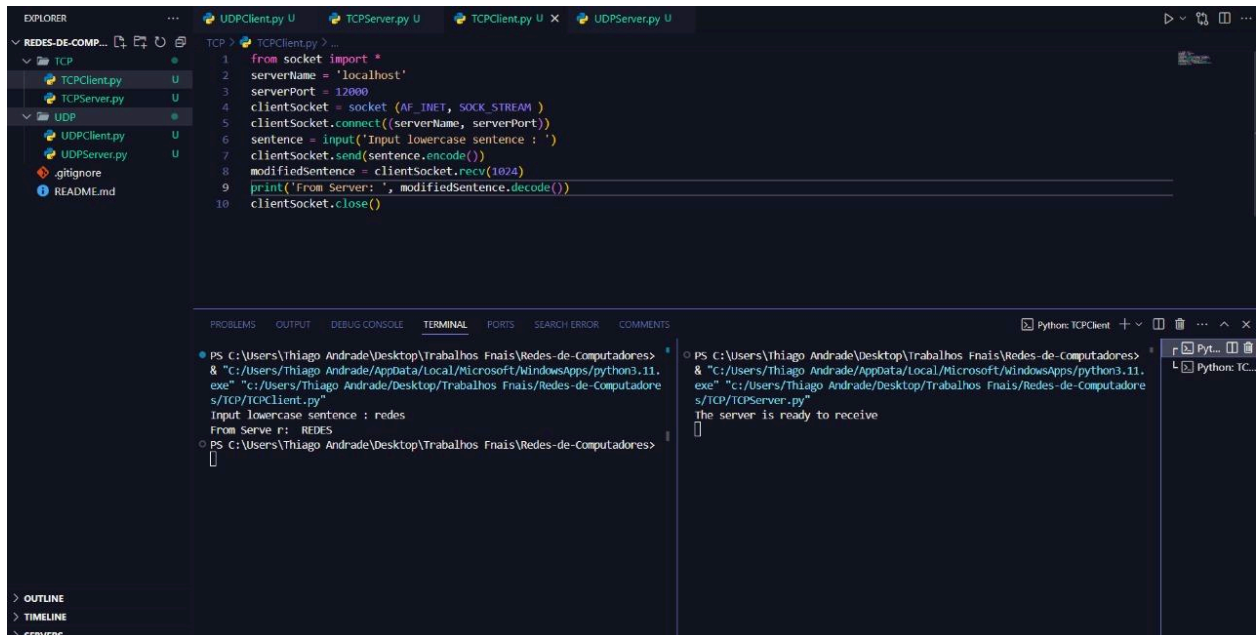

# Depois de enviar a sentença modificada ao cliente, fechamos o
socket da

# conexão. Mas como o serverSocket permanece aberto, outro
cliente agora pode bater à

# porta e enviar uma sentença ao servidor, para que seja
modificada

connectionSocket.close()
```

6. TCP - Server Rodando



```
EXPLORER
REDES-DE-COMP...
  TCP
    TCPClient.py
    TCPServer.py
  UDP
    UDPClient.py
    UDPServer.py
  .gitignore
  README.md

TCPClient.py
1 from socket import *
2 serverName = 'localhost'
3 serverPort = 12000
4 clientSocket = socket(AF_INET, SOCK_STREAM)
5 clientSocket.connect((serverName, serverPort))
6 sentence = input('Input lowercase sentence : ')
7 clientSocket.send(sentence.encode())
8 modifiedSentence = clientSocket.recv(1024)
9 print('From Server: ', modifiedSentence.decode())
10 clientSocket.close()

TCPServer.py
1 from socket import *
2 serverName = 'localhost'
3 serverPort = 12000
4 serverSocket = socket(AF_INET, SOCK_STREAM)
5 serverSocket.bind((serverName, serverPort))
6 serverSocket.listen(1)
7 print('The server is ready to receive')
8 (clientSocket, address) = serverSocket.accept()
9 print('Received from', address)
10 sentence = clientSocket.recv(1024)
11 modifiedSentence = sentence.decode().upper()
12 clientSocket.send(modifiedSentence.encode())
13 clientSocket.close()

TERMINAL
PS C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores>
& "C:\Users\Thiago Andrade\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores\TCP\TCPClient.py"
Input lowercase sentence : redes
From Server: REDES
PS C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores>
```

3. Execução/captura usando o cliente netcat

TCP:

1. Primeiramente baixei o netcat para Windows, e extrai o arquivo zip do mesmo.
2. Em seguida, iniciei o servidor TCP passando o seguinte comando para o Prompt de Comando: “python "C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores\TCP\TCPServer.py”.
3. Após iniciar o servidor, abri uma nova janela do Prompt de Comando para conectar o netcat ao servidor, e passei o seguinte comando: “nc 127.0.0.1 12000”. Os parâmetros, nesse caso, “nc” é usado para executar o netcat, o “127.0.0.1” é usado para passar o endereço IP do servidor, o localhost e por fim o “12000” para passar a porta na qual o servidor está escutando, já definida no próprio código anteriormente.
4. Com a conexão TCP estabelecida, finalmente conseguirei enviar os dados e capturar a resposta, e assim digitei “redes”, e o servidor converteu essas letras para maiúsculas, retornando “REDES”.

Conclusão

O teste com o netcat foi bem-sucedido. Ele se mostrou uma ferramenta eficaz e

simples para estabelecer conexões TCP, sem a necessidade de programação adicional, facilitando o processo de verificação do funcionamento correto do servidor. O uso do netcat é uma forma rápida e eficiente de validar a comunicação de rede em um ambiente controlado.

The screenshot displays a Windows development environment with three main components: a file explorer, a code editor, and a command prompt.

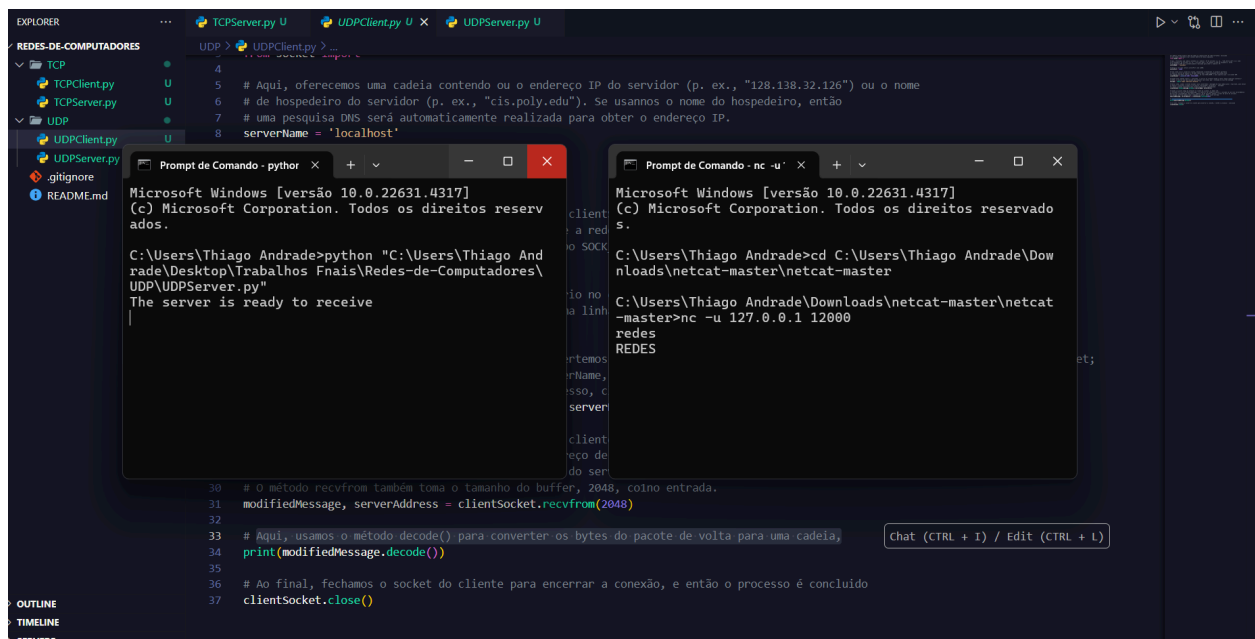
- File Explorer:** Shows a project structure with folders for 'TCP' and 'UDP'. The 'TCP' folder contains 'TCPClient.py' and 'TCPServer.py'. The 'UDP' folder contains 'UDPCClient.py' and 'UDPServer.py'.
- Code Editor:** Displays the 'TCPServer.py' file. The code includes a comment in Portuguese explaining the use of the 'socket' module and a Python script that sets up a TCP server, listens for connections, and prints the received message in uppercase.
- Command Prompt:** Shows the execution of the server and client. The server is started with the command: `python "C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores\TCP\TCPServer.py"`. The output shows 'The server is ready to receive'. Then, a netcat client is used to connect to the server: `nc 127.0.0.1 12000`. The client sends the word 'redes', and the server responds with 'REDES'.

UDP:

1. Primeiramente baixei o netcat para Windows, e extrai o arquivo zip do mesmo.
2. Em seguida, iniciei o servidor UDP passando o seguinte comando para o Prompt de Comando: “python "C:\Users\Thiago Andrade\Desktop\Trabalhos Finais\Redes-de-Computadores\UDP\UDPServer.py"”.
3. Após iniciar o servidor, abri uma nova janela do Prompt de Comando para conectar o netcat ao servidor, e passei o seguinte comando: “nc -u 127.0.0.1 12000”. Os parâmetros, nesse caso, “nc” é usado para executar o netcat, o “-u” informa ao netcat que deve usar o protocolo UDP, o “127.0.0.1” é usado para passar o endereço IP do servidor, o localhost, por fim o “12000” para passar a porta na qual o servidor está escutando, já definida no próprio código anteriormente.
4. Com a conexão UDP estabelecida, finalmente conseguirei enviar os dados e capturar a resposta, e assim digitei “redes”, e o servidor converteu essas letras para maiúsculas, retornando “REDES”.

Conclusão

O uso do netcat como cliente UDP foi simples e eficaz para testar a comunicação entre cliente e servidor. Com o comando `-u`, conseguimos ajustar o netcat para funcionar com o protocolo UDP, permitindo a troca de mensagens de maneira rápida e sem complicações. A ferramenta mostrou sua versatilidade, funcionando tanto com conexões TCP quanto UDP, o que facilita muito a verificação de redes. Esse teste também ajudou a entender melhor como o UDP funciona, destacando a diferença em relação ao TCP, já que o UDP não depende de uma conexão estabelecida para a troca de dados.



The screenshot displays a Windows desktop environment. On the left, a File Explorer window shows a directory structure with folders for 'TCP' and 'UDP', each containing 'TCPClient.py' and 'UDPServer.py'. The 'UDP' folder is selected. In the center, a 'Prompt de Comando - python' window shows the execution of a Python script: `C:\Users\Thiago Andrade>python "C:\Users\Thiago Andrade\Desktop\Trabalhos Fnais\Redes-de-Computadores\UDP\UDPServer.py"`, with the output 'The server is ready to receive'. On the right, a 'Prompt de Comando - nc -u' window shows the netcat master process being started: `C:\Users\Thiago Andrade>cd C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master` and `-master>nc -u 127.0.0.1 12000`, followed by the receipt of a message: `redes`.

4. Execução/captura usando o server netcat

TCP:

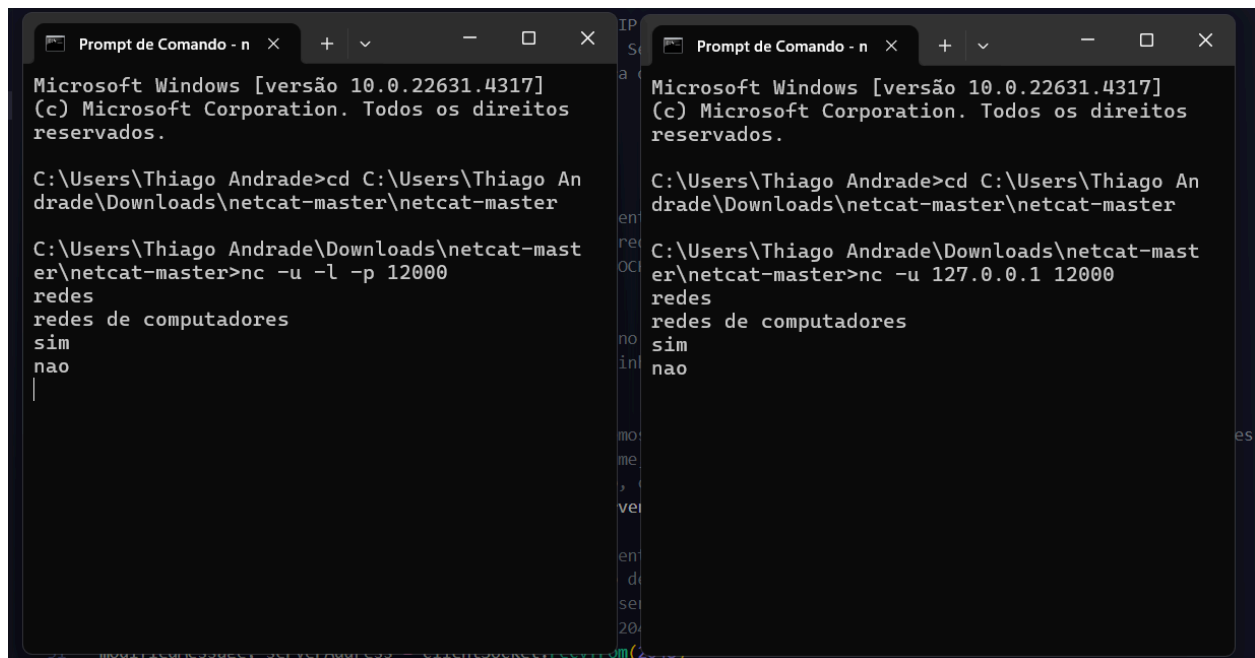
1. Primeiramente baixei o netcat para Windows, e extrai o arquivo zip do mesmo.
2. Em seguida, iniciei o servidor TCP passando o seguinte comando para o Prompt de Comando: `"nc -l -p 12000"`. Os parâmetros, nesse caso, `"nc"` é usado para executar o netcat, o `"-l"` coloca o netcat em modo de escuta, ou seja, ele fica esperando por conexões de clientes, o `"-p 12000"` é usado para definir a porta onde o servidor vai escutar, neste caso, a porta 12000.
3. Após iniciar o servidor, abri uma nova janela do Prompt de Comando para

conectar o netcat ao servidor, e passei o seguinte comando: “nc 127.0.0.1 12000”. Os parâmetros, nesse caso, “nc” é usado para executar o netcat, o “127.0.0.1” é usado para passar o endereço IP do servidor, o localhost e por fim o “12000” para passar a porta na qual o servidor está escutando, já definida no próprio código anteriormente.

4. Com a conexão TCP estabelecida, finalmente conseguirei enviar os dados e capturar a resposta.

Conclusão

Usando o netcat como servidor TCP foi possível notar que a conversão de mensagens para letras maiúsculas não foi realizada. A razão para isso é que o netcat não altera mensagens recebidas e as envia de volta. Na implementação em Python, há uma linha na função que converte as mensagens recebidas para letras maiúsculas antes de enviá-las de volta ao cliente. Além disso, o netcat não oferece essa funcionalidade, ele recebe os dados e os transmite exatamente como estão. Portanto, quando o cliente envia a mensagem, o servidor netcat envia a mensagem exatamente sem modificá-la e, portanto, a mensagem permanece em letras minúsculas. Isso significa que, se quisermos enviar ou receber e depois processar ou ajustar os dados enviados, precisamos de uma lógica. Então se quisermos usar o netcat para enviar mensagens em tempo real e alterar esses dados enviados, precisamos de uma lógica por trás para fazer isso.



```
Prompt de Comando - n x + - □ x IP Sc
Microsoft Windows [versão 10.0.22631.4317]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Thiago Andrade>cd C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master

C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master>nc -u -l -p 12000
redes
redes de computadores
sim
nao
|

Prompt de Comando - n x + - □ x IP Sc
Microsoft Windows [versão 10.0.22631.4317]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Thiago Andrade>cd C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master

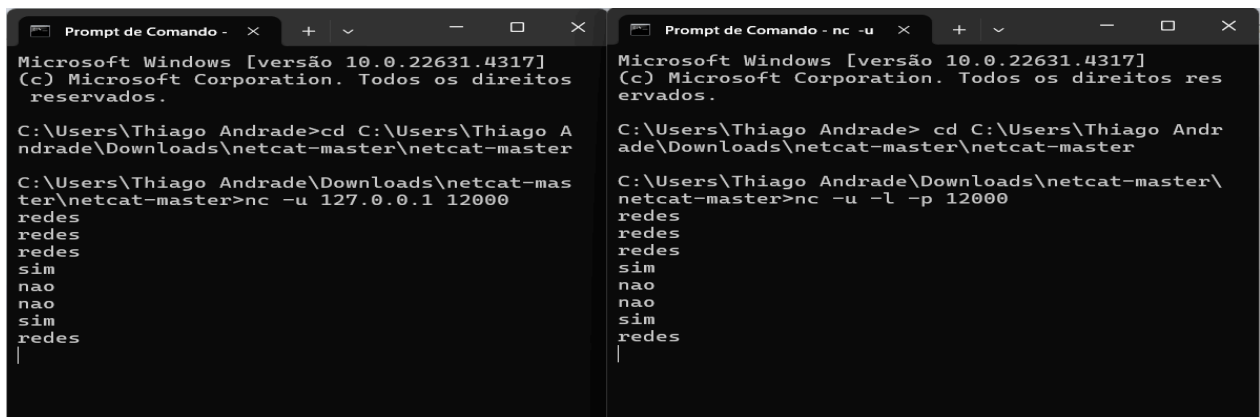
C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master>nc -u 127.0.0.1 12000
redes
redes de computadores
sim
nao
```

UDP:

1. Primeiramente baixei o netcat para Windows, e extrai o arquivo zip do mesmo.
2. Em seguida, iniciei o servidor UTP passando o seguinte comando para o Prompt de Comando: “nc -u -l -p 12000”. Os parâmetros, nesse caso, “nc” é usado para executar o netcat, o “-u” indica que o netcat deve usar o protocolo UDP, o “-l” coloca o netcat em modo de escuta, ou seja, ele fica esperando por conexões de clientes, o “-p 12000” é usado para definir a porta onde o servidor vai escutar, neste caso, a porta 12000.
3. Após iniciar o servidor, abri uma nova janela do Prompt de Comando para conectar o netcat ao servidor, e passei o seguinte comando: “nc -u 127.0.0.1 12000”. Os parâmetros, nesse caso, “nc” é usado para executar o netcat, o “-u” informa ao netcat que deve usar o protocolo UDP, o “127.0.0.1” é usado para passar o endereço IP do servidor, o localhost, por fim o “12000” para passar a porta na qual o servidor está escutando, já definida no próprio código anteriormente.
4. Com a conexão UTP estabelecida, finalmente conseguirei enviar os dados e capturar as respostas.

Conclusão

Usar o netcat como servidor UDP, vemos que, o resultado é igual ao teste de TCP, a as mensagens continuam sem converter para maiúsculas. Isso ocorre porque o netcat, basicamente, serviu apenas para retransmitir a mensagem recebida. Dessa maneira, enquanto a implementação de Python converte a mensagem recebida no servidor e retransmite para o cliente em maiúsculas, o netcat não tem para onde enviar já que não possui nenhuma lógica. Quando isso é feito, a saída permanece da maneira que foi originalmente, sem nenhuma alteração.



```
Microsoft Windows [versão 10.0.22631.4317]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Thiago Andrade>cd C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master

C:\Users\Thiago Andrade\Downloads\netcat-master>nc -u 127.0.0.1 12000
redes
redes
redes
sim
nao
nao
sim
redes
|

Microsoft Windows [versão 10.0.22631.4317]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Thiago Andrade> cd C:\Users\Thiago Andrade\Downloads\netcat-master\netcat-master

C:\Users\Thiago Andrade\Downloads\netcat-master>ncat-master>nc -u -l -p 12000
redes
redes
redes
sim
nao
nao
sim
redes
|
```

5. CONCLUSÃO FINAL

Este relatório sobre a comunicação de rede usando sockets, tanto para UDP quanto para TCP, ajudou a entender melhor como funciona a troca de dados entre cliente e servidor. Através dos exemplos do livro "Redes de Computadores e a Internet" de Kurose e Ross, foi possível ver como as mensagens são enviadas e recebidas, além das principais diferenças entre os protocolos TCP e UDP. O uso do netcat como cliente e servidor mostrou ser uma ferramenta prática para simular conexões de rede e testar a comunicação de forma rápida, sem precisar de muito código. No entanto, ficou claro que o netcat não faz nenhuma alteração nas mensagens, o que destaca a importância de incluir lógica adicional nas aplicações que precisam processar informações, como converter texto. Em resumo, este trabalho enfatiza não apenas a funcionalidade dos sockets, mas também a necessidade de desenvolver implementações que atendam a requisitos específicos de manipulação de dados nas comunicações de rede.

6. REFERÊNCIAS

1. **Kurose, J. F., & Ross, K. W. (2018).** *Redes de Computadores e a Internet* (8ª ed.). Pearson.
2. **Varonis. (n.d.).** *Netcat commands*. Recuperado em 15/10/2024, de <https://www.varonis.com/pt-br/blog/netcat-commands>
3. **Diego C. (n.d.).** *Netcat*. Recuperado em 15/10/2024, de <https://github.com/diegocr/netcat>