

# Desenvolvimento de protocolo de aplicação para Chat

Guilherme Argilar

[guilherme.argilar@eud.pucrs.br](mailto:guilherme.argilar@eud.pucrs.br)

## 1. Aplicação Implementada

Este relatório aborda a implementação de clientes e servidores utilizando os protocolos de comunicação de TCP e UDP em Python, enfatizando similaridades fundamentais entre essas abordagens de rede. Mesmo que os protocolos de TCP e UDP sejam estruturalmente indicados para garantia de entrega e ordem de pacotes, as implementações compartilhadas compartilham vários aspectos estruturais e funcionais. Em ambas as implementações, o módulo de soquete do Python é fundamental na programação de rede que podem ser aplicados de uma forma fortemente relacionada a vários sistemas de comunicação. O relatório discute ainda o uso de threads para gerenciar várias conexões de clientes no servidor e a separação de entrada/saída de processo no cliente para facilitar a interação contínua e bidirecional.

### 1.1 Implementação para o protocolo TCP

Esta seção tem como objetivo explorar o processo de decisão adotado para a implementação do cliente e servidor que operam sob o protocolo TCP.

#### 1.1.1 Implementação do servidor TCP

Para iniciar o servidor, utilizamos a biblioteca socket configurada no modo `SOCK_STREAM`, que basicamente prepara o servidor para transmitir dados em um fluxo contínuo. Ele fica ouvindo em uma porta específica que definimos para esperar as conexões dos clientes. Quando um cliente tenta se conectar, o servidor aceita essa conexão através do método `accept()`, criando um novo socket específico para essa conexão.

Para lidar com várias conexões de forma eficiente foi utilizado threads. Cada conexão é gerenciada por uma thread diferente, permitindo assim que o servidor continue aceitando novas conexões, e, ao mesmo tempo, gerencie o tráfego de dados das conexões já estabelecidas.

No que diz respeito ao envio de arquivos, foi implementada uma lógica específica no nosso servidor TCP para tratar essas solicitações.

#### 1.1.2 Implementação do cliente TCP

A configuração do cliente TCP começa com a criação de um socket que utiliza o protocolo TCP para estabelecer uma conexão confiável com um servidor específico. O processo envolve o uso do método `connect()`, que sincroniza o cliente e o servidor para permitir uma comunicação bidirecional.

No lado do cliente, a interação é principalmente gerenciada por uma interface que lida com a entrada do usuário. Comandos específicos são processados e enviados ao servidor. Uma característica crucial desta implementação é a utilização de threads, que permitem que o cliente execute tarefas de recepção de dados em paralelo com a entrada do usuário.

O encerramento da conexão é tratado para garantir que todas as comunicações sejam concluídas antes do fechamento do socket. O comando */QUIT* é um exemplo de como os clientes podem encerrar a conexão de forma ordenada.

## 1.2 Implementação para o protocolo UDP

Esta seção tem como objetivo explorar o processo de decisão adotado para a implementação do cliente e servidor que operam sob o protocolo UDP.

### 1.2.1 Implementação do servidor UDP

Na implementação do servidor UDP, o socket é criado com o tipo *SOCK\_DGRAM*, para transmissões de datagrama. A principal função deste tipo de servidor é escutar em um endereço IP e porta especificados usando o método *bind()*, o que permite ao servidor receber pacotes de qualquer cliente sem a necessidade de estabelecer uma conexão persistente.

Dentro da implementação, o servidor UDP lida com a entrada de dados utilizando *recvfrom()*, que também fornece a origem dos dados. Isso permite ao servidor responder diretamente ao endereço de origem, utilizando o método *sendto()*, facilitando a comunicação bidirecional, ainda que de forma independente para cada pacote. A natureza stateless do UDP significa que o servidor não mantém um estado de conexão, permitindo que ele gerencie as solicitações de vários clientes de maneira mais simples e direta.

Quando o comando para enviar um arquivo é recebido, o servidor extrai informações como o destinatário, o nome do arquivo e seu tamanho do comando recebido. O servidor então aguarda a recepção do arquivo inteiro e, uma vez recebido, envia os dados diretamente ao destinatário especificado, se presente no registro de usuários ativos. Esse processo é efetuado sem a necessidade de dividir o arquivo em múltiplos pacotes, aproveitando a capacidade do UDP de enviar grandes blocos de dados em um único envio, desde que dentro do limite de tamanho de um datagrama UDP.

### 1.2.2 Implementação do cliente UDP

O cliente UDP inicia sua operação criando um socket através do comando *socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)*. Este socket é utilizado para enviar e receber dados do servidor, identificado pelo seu endereço IP e porta.

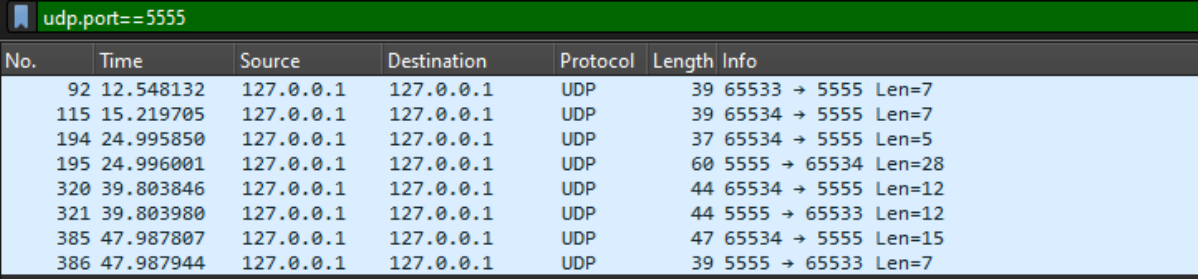
Na parte de recebimento, o cliente mantém um loop ativo dentro de uma thread separada, evitando bloquear a interação do usuário.

Para transmitir um arquivo, o cliente primeiro prepara os dados do arquivo, lendo todo o conteúdo na memória com *file.read()*. Após isso, um cabeçalho contendo o comando */FILE*, o destinatário, o nome e o tamanho do arquivo é enviado ao servidor em um pacote único através do *client\_socket.sendto(header, server\_address)*, seguido pelo envio dos dados do arquivo.

Quando o usuário decide encerrar a sessão, um comando */QUIT* é enviado ao servidor para notificar a desconexão. O socket é então fechado e a *thread* de recebimento é finalizada, garantindo a liberação dos recursos.

## 2. Análise de Tráfego

1) Execute o Wireshark para monitorar o tráfego UDP gerado pelo programa. Identifique os pacotes UDP que estão sendo enviados para cada um dos servidores. Quais portas de origem e destino estão sendo utilizadas pelos pacotes?



No.	Time	Source	Destination	Protocol	Length	Info
92	12.548132	127.0.0.1	127.0.0.1	UDP	39	65533 → 5555 Len=7
115	15.219705	127.0.0.1	127.0.0.1	UDP	39	65534 → 5555 Len=7
194	24.995850	127.0.0.1	127.0.0.1	UDP	37	65534 → 5555 Len=5
195	24.996001	127.0.0.1	127.0.0.1	UDP	60	5555 → 65534 Len=28
320	39.803846	127.0.0.1	127.0.0.1	UDP	44	65534 → 5555 Len=12
321	39.803980	127.0.0.1	127.0.0.1	UDP	44	5555 → 65533 Len=12
385	47.987807	127.0.0.1	127.0.0.1	UDP	47	65534 → 5555 Len=15
386	47.987944	127.0.0.1	127.0.0.1	UDP	39	5555 → 65533 Len=7

Wireshark da aplicação UDP

pacote Nº 92: /REG  
pacote Nº 115: /REG  
pacote Nº 194: /LIST  
pacote Nº 195: Retorno do /LIST  
pacote Nº 320: /S  
pacote Nº 321: /RECEBIMENTO DO /S pelo o outro cliente  
pacote Nº 385: /MSG  
pacote Nº 386: RECEBIMENTO DO /MSG pelo o outro cliente

A portas de origem e destinos utilizadas são:

5555 - servidor  
65533 - cliente 1  
65534 - cliente 2

### 2) Há diferença, em termos de volume de tráfego na rede, entre a aplicação com socket TCP e a aplicação com socket UDP?

Ao fazer os mesmos comandos, podemos calcular que o volume total de tráfego para a aplicação que utiliza o protocolo TCP é de 1.110 bytes, enquanto para a aplicação que utiliza o protocolo UDP é de 349 bytes. Observa-se, portanto, uma diferença significativa no volume de tráfego entre as duas aplicações, sendo que a aplicação TCP gera mais tráfego. Essa diferença pode ser atribuída às características distintas de cada protocolo. O TCP, por ser um protocolo orientado à conexão, inclui uma maior sobrecarga devido à necessidade de estabelecer uma conexão, além de mecanismos de controle de congestionamento e

garantias de entrega de dados. Em contraste, o UDP, por ser um protocolo sem conexão, envolve menos sobrecarga, o que resulta em um volume de tráfego menor.

### 3 Há diferença, em termos de desempenho da aplicação, entre a aplicação com socket TCP e a aplicação com socket UDP?

Em uma aplicação de chat de menor escala, que priorize eficiência e simplicidade ao invés de segurança rigorosa e integridade absoluta de dados, o protocolo UDP pode ser mais vantajoso que o TCP. No entanto, se houver riscos de instabilidade na rede, o TCP torna-se a opção preferível, pois garante que a integridade dos dados da conversa seja preservada.

### 4) Compare a transmissão de um arquivo de 1200 bytes usando a socket TCP e socket UDP.

No.	Time	Source	Destination	Protocol	Length	Info
43	5.742738	127.0.0.1	127.0.0.1	TCP	56	64921 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
44	5.742779	127.0.0.1	127.0.0.1	TCP	56	5555 → 64921 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
45	5.742795	127.0.0.1	127.0.0.1	TCP	44	64921 → 5555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
52	7.132648	127.0.0.1	127.0.0.1	TCP	51	64921 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
53	7.132663	127.0.0.1	127.0.0.1	TCP	44	5555 → 64921 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
98	10.411127	127.0.0.1	127.0.0.1	TCP	56	64928 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
99	10.411167	127.0.0.1	127.0.0.1	TCP	56	5555 → 64928 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
100	10.411184	127.0.0.1	127.0.0.1	TCP	44	64928 → 5555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
122	13.612715	127.0.0.1	127.0.0.1	TCP	51	64928 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
123	13.612732	127.0.0.1	127.0.0.1	TCP	44	5555 → 64928 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
249	29.205237	127.0.0.1	127.0.0.1	TCP	2082	64921 → 5555 [PSH, ACK] Seq=8 Ack=1 Win=2619648 Len=2038
250	29.205254	127.0.0.1	127.0.0.1	TCP	44	5555 → 64921 [ACK] Seq=1 Ack=2046 Win=2617600 Len=0
251	29.205293	127.0.0.1	127.0.0.1	TCP	2079	5555 → 64928 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=2035
252	29.205305	127.0.0.1	127.0.0.1	TCP	44	64928 → 5555 [ACK] Seq=8 Ack=2036 Win=2617600 Len=0

#### Wireshark da transmissão do arquivo com 1200 bytes pela aplicação TCP

No.	Time	Source	Destination	Protocol	Length	Info
829	11.118014	127.0.0.1	127.0.0.1	UDP	39	50094 → 5555 Len=7
1019	13.726094	127.0.0.1	127.0.0.1	UDP	39	55791 → 5555 Len=7
2358	32.454167	127.0.0.1	127.0.0.1	UDP	74	50094 → 5555 Len=42
2359	32.454214	127.0.0.1	127.0.0.1	UDP	1232	50094 → 5555 Len=1200
2360	32.454442	127.0.0.1	127.0.0.1	UDP	71	5555 → 55791 Len=39
2361	32.454471	127.0.0.1	127.0.0.1	UDP	1232	5555 → 55791 Len=1200

#### Wireshark da transmissão do arquivo com 1200 bytes pela aplicação UDP

Ao comparar a transmissão de um arquivo de 1200 bytes usando TCP e UDP, percebemos diferenças claras na eficiência e no comportamento dos protocolos. O TCP é mais confiável, confirmando cada pacote recebido e retransmitindo os perdidos, mas isso resulta em maior overhead devido às etapas de conexão e transmissão de cabeçalhos, tornando a transmissão mais lenta e volumosa.

Já o UDP é mais eficiente, operando sem estabelecer conexão, o que reduz o overhead e acelera a transmissão em redes estáveis. No entanto, ele não garante a entrega ou a ordem dos pacotes, e pacotes perdidos não são retransmitidos a menos que aplicativos específicos implementem correções.

### 5) Compare a transmissão de um arquivo de 2000 bytes usando a socket TCP e socket UDP.

No.	Time	Source	Destination	Protocol	Length	Info
113	14.867381	127.0.0.1	127.0.0.1	TCP	56	49814 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
114	14.867422	127.0.0.1	127.0.0.1	TCP	56	5555 → 49814 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
115	14.867438	127.0.0.1	127.0.0.1	TCP	44	49814 → 5555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
137	17.489486	127.0.0.1	127.0.0.1	TCP	51	49814 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
138	17.489502	127.0.0.1	127.0.0.1	TCP	44	5555 → 49814 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
192	25.353513	127.0.0.1	127.0.0.1	TCP	56	49824 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
193	25.353550	127.0.0.1	127.0.0.1	TCP	56	5555 → 49824 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
194	25.353569	127.0.0.1	127.0.0.1	TCP	44	49824 → 5555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
216	27.001625	127.0.0.1	127.0.0.1	TCP	51	49824 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
217	27.001640	127.0.0.1	127.0.0.1	TCP	44	5555 → 49824 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
337	41.633794	127.0.0.1	127.0.0.1	TCP	2082	49814 → 5555 [PSH, ACK] Seq=8 Ack=1 Win=2619648 Len=2038
338	41.633835	127.0.0.1	127.0.0.1	TCP	44	5555 → 49814 [ACK] Seq=1 Ack=2046 Win=2617600 Len=0
339	41.633847	127.0.0.1	127.0.0.1	TCP	2079	5555 → 49824 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=2035
340	41.633861	127.0.0.1	127.0.0.1	TCP	44	49824 → 5555 [ACK] Seq=8 Ack=2036 Win=2617600 Len=0

### Wireshark da transmissão do arquivo com 2000 bytes pela aplicação TCP

No.	Time	Source	Destination	Protocol	Length	Info
201	2.729510	127.0.0.1	127.0.0.1	UDP	39	61377 → 5555 Len=7
780	10.401494	127.0.0.1	127.0.0.1	UDP	39	62957 → 5555 Len=7
1567	21.313541	127.0.0.1	127.0.0.1	UDP	74	61377 → 5555 Len=42
1568	21.313584	127.0.0.1	127.0.0.1	UDP	2032	61377 → 5555 Len=2000
1569	21.313800	127.0.0.1	127.0.0.1	UDP	71	5555 → 62957 Len=39
1570	21.313828	127.0.0.1	127.0.0.1	UDP	2032	5555 → 62957 Len=2000

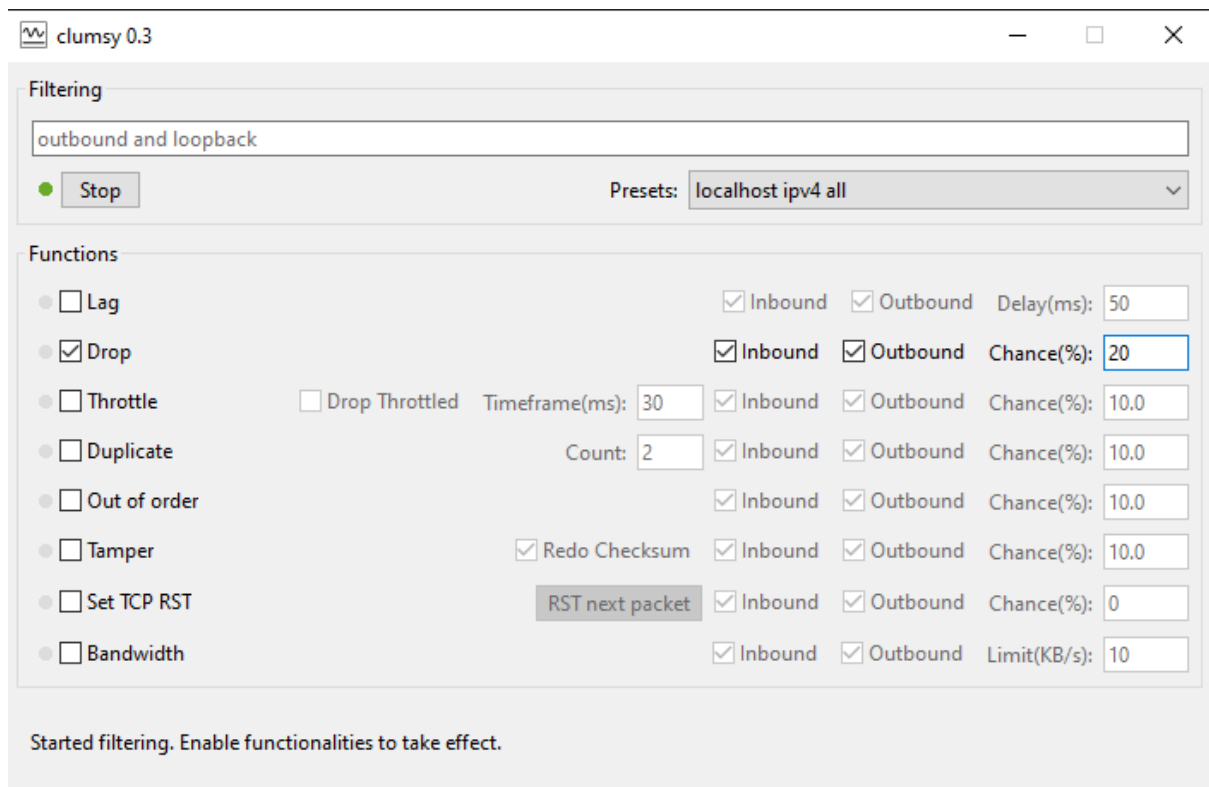
### Wireshark da transmissão do arquivo com 2000 bytes pela aplicação UDP

## 6) Configure a interface de rede da máquina para incluir perda de pacotes.

a. Qual a diferença, em termos de tráfego na rede, entre o socket TCP e UDP?

Houve alguma retransmissão usando TCP?

Para isso utilizei o software Clumsy, com 20% de perda de pacote.



### Clumsy

Quando ocorre perda de pacote, a diferença em termos de tráfego reside no fato de que o TCP, sendo um protocolo orientado a conexões, realiza uma confirmação em três etapas (handshake). Se houver perda de pacote durante a transmissão, o pacote é reenviado. Em contraste, o UDP não realiza essa confirmação, portanto, é como se o pacote perdido nunca tivesse existido.

No exemplo de aplicação usando UDP, um cliente com identificação 12 (porta 63163) está enviando uma série de mensagens em um chat. Contudo, ao tentar enviar a mensagem número 4, ocorreu uma perda de pacote e ela não foi registrada. Enquanto isso, o cliente 34 (porta 61025), que aguardava as mensagens do cliente 12, recebeu apenas as mensagens 3 e 5, uma vez que também houve perda de pacotes contendo as mensagens 1 e 2 durante a comunicação do servidor para o cliente 34.

```
Digite seu apelido para registro: 12
-> /S MENSAGEM 1
-> /S MENSAGEM 2
-> /S MENSAGEM 3
-> /S MENSAGEM 4
-> /S MENSAGEM 5
-> 
```

**Envio de mensagens do cliente 12**

```
Digite seu apelido para registro: 34
-> 12: /S MENSAGEM 3
12: /S MENSAGEM 5

```

**Recebimento de mensagens do cliente 34**

```
Servidor iniciado e ouvindo em UDP no porto 5555.
Dado recebido de ('127.0.0.1', 63163): b'/REG 12'
12 registrado do endereço ('127.0.0.1', 63163).
Dado recebido de ('127.0.0.1', 61025): b'/REG 34'
34 registrado do endereço ('127.0.0.1', 61025).
Dado recebido de ('127.0.0.1', 63163): b'12: /S MENSAGEM 1'
Dado recebido de ('127.0.0.1', 63163): b'12: /S MENSAGEM 2'
Dado recebido de ('127.0.0.1', 63163): b'12: /S MENSAGEM 3'

```

**Servidor**

udp.port==5555							
No.	Time	Source	Destination	Protocol	Length	Info	
67	5.856594	127.0.0.1	127.0.0.1	UDP	39	63163 → 5555	Len=7
1046	93.768674	127.0.0.1	127.0.0.1	UDP	39	61025 → 5555	Len=7
1181	108.944852	127.0.0.1	127.0.0.1	UDP	49	63163 → 5555	Len=17
1196	112.608379	127.0.0.1	127.0.0.1	UDP	49	63163 → 5555	Len=17
1203	114.536435	127.0.0.1	127.0.0.1	UDP	49	63163 → 5555	Len=17
1204	114.536622	127.0.0.1	127.0.0.1	UDP	49	5555 → 61025	Len=17
4243	374.624666	127.0.0.1	127.0.0.1	UDP	49	63163 → 5555	Len=17
4244	374.624898	127.0.0.1	127.0.0.1	UDP	49	5555 → 61025	Len=17

### Wireshark da aplicação com 20% de perda de pacotes

Houve 3 retransmissões na aplicação TCP, uma logo no registro do primeiro cliente, e outras duas durante a troca de mensagens entre os clientes.

tcp.port==5555							
No.	Time	Source	Destination	Protocol	Length	Info	
43	5.506868	127.0.0.1	127.0.0.1	TCP	56	52294 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
44	5.506971	127.0.0.1	127.0.0.1	TCP	56	5555 → 52294 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
45	5.507088	127.0.0.1	127.0.0.1	TCP	44	52294 → 5555 [ACK] Seq=1 Ack=1 Win=2619648 Len=0	
79	7.775101	127.0.0.1	127.0.0.1	TCP	51	52294 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7	
80	8.078303	127.0.0.1	127.0.0.1	TCP	51	[TCP Retransmission] 52294 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7	
61	8.078344	127.0.0.1	127.0.0.1	TCP	56	5555 → 52294 [ACK] Seq=1 Ack=8 Win=2619648 Len=0 SLE=1 SRE=8	
143	14.843871	127.0.0.1	127.0.0.1	TCP	56	52304 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
179	17.874563	127.0.0.1	127.0.0.1	TCP	56	5555 → 52304 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
214	21.687098	127.0.0.1	127.0.0.1	TCP	51	52304 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=7	
215	21.687145	127.0.0.1	127.0.0.1	TCP	44	5555 → 52304 [ACK] Seq=1 Ack=8 Win=327424 Len=0	
287	30.340761	127.0.0.1	127.0.0.1	TCP	54	52304 → 5555 [PSH, ACK] Seq=8 Ack=1 Win=327424 Len=10	
288	30.340874	127.0.0.1	127.0.0.1	TCP	51	5555 → 52294 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=7	
290	30.640842	127.0.0.1	127.0.0.1	TCP	51	[TCP Retransmission] 5555 → 52294 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=7	
291	30.640889	127.0.0.1	127.0.0.1	TCP	56	52294 → 5555 [ACK] Seq=8 Ack=8 Win=2619648 Len=0 SLE=1 SRE=8	
294	30.941892	127.0.0.1	127.0.0.1	TCP	54	[TCP Retransmission] 52304 → 5555 [PSH, ACK] Seq=8 Ack=1 Win=327424 Len=10	
295	30.941936	127.0.0.1	127.0.0.1	TCP	56	5555 → 52304 [ACK] Seq=1 Ack=18 Win=327424 Len=0 SLE=8 SRE=18	

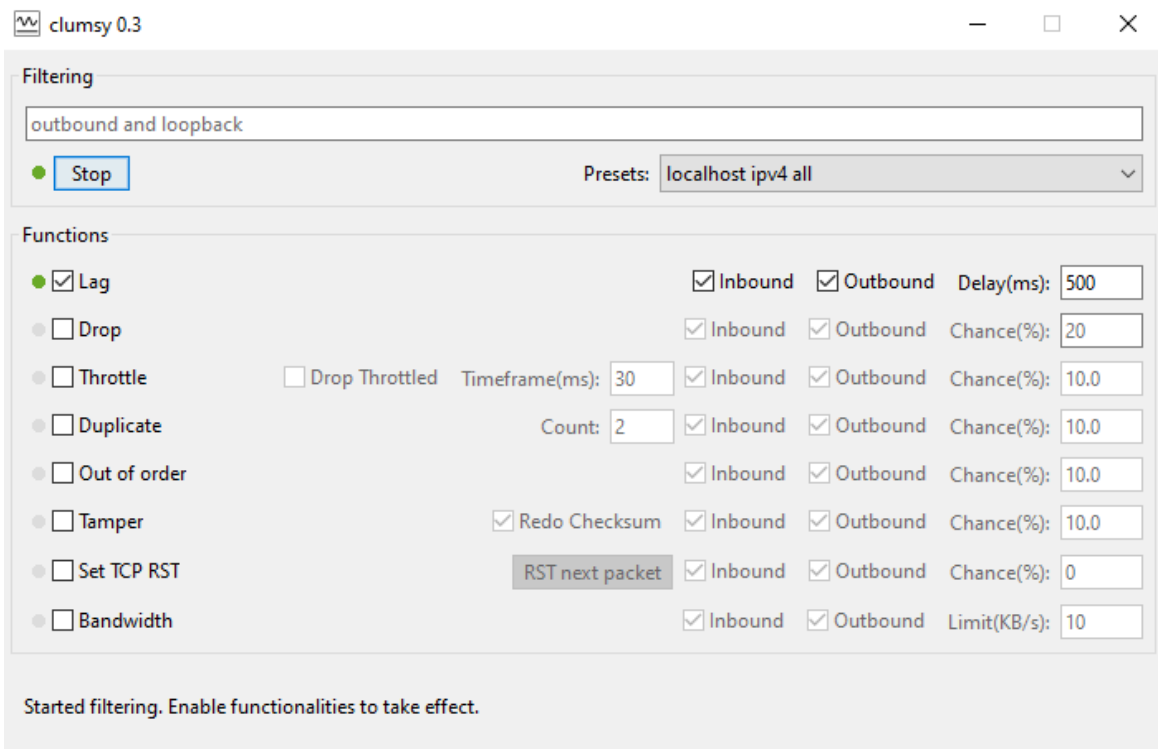
### Wireshark da aplicação TCP

7) Configurar a interface de rede da máquina para incluir latência variável.

a. Qual a diferença, em termos de tráfego na rede, entre o socket TCP e UDP?

Houve alguma retransmissão usando TCP?





## Configuração do Clumsy para a latência

O tráfego TCP pode variar significativamente com a introdução de latência variável devido às suas retransmissões e ajustes dinâmicos na taxa de transmissão, visando acomodar as mudanças nas condições da rede. Por outro lado, o tráfego UDP permanece constante, pois não possui mecanismos para ajustar ou retransmitir baseado nas condições de latência, o que pode resultar em perda de dados ou entrega fora de ordem sem que haja qualquer compensação pela aplicação ou protocolo.

Porém, como visto nas imagens não foi possível demonstrar este fato na aplicação UDP. Já a aplicação TCP, fez os ajustes necessários quando houve problema por conta da latência.

tcp.port==5555					
No.	Time	Source	Destination	Protocol	Length Info
849	38.058687	127.0.0.1	127.0.0.1	TCP	56 57501 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
858	38.562834	127.0.0.1	127.0.0.1	TCP	56 5555 → 57501 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
864	39.071907	127.0.0.1	127.0.0.1	TCP	56 [TCP Retransmission] 57501 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
866	39.071963	127.0.0.1	127.0.0.1	TCP	44 57501 → 5555 [ACK] Seq=1 Ack=1 Win=327424 Len=0
884	39.583032	127.0.0.1	127.0.0.1	TCP	56 [TCP Retransmission] 5555 → 57501 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
913	40.096194	127.0.0.1	127.0.0.1	TCP	56 [TCP Dup ACK 866#1] 57501 → 5555 [ACK] Seq=1 Ack=1 Win=327424 Len=0 SLE=0 SRE=1
971	43.061380	127.0.0.1	127.0.0.1	TCP	51 57501 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=7
975	43.590510	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=8 Win=327424 Len=0
1086	49.180085	127.0.0.1	127.0.0.1	TCP	56 57515 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1097	49.711201	127.0.0.1	127.0.0.1	TCP	56 5555 → 57515 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1110	50.109300	127.0.0.1	127.0.0.1	TCP	56 [TCP Retransmission] 57515 → 5555 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1111	50.227303	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=1 Ack=1 Win=327424 Len=0
1128	50.733400	127.0.0.1	127.0.0.1	TCP	56 [TCP Retransmission] 5555 → 57515 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1152	51.260542	127.0.0.1	127.0.0.1	TCP	56 [TCP Dup ACK 1111#1] 57515 → 5555 [ACK] Seq=1 Ack=1 Win=327424 Len=0 SLE=0 SRE=1
1166	52.119737	127.0.0.1	127.0.0.1	TCP	51 57515 → 5555 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=7
1176	52.652860	127.0.0.1	127.0.0.1	TCP	44 5555 → 57515 [ACK] Seq=1 Ack=8 Win=327424 Len=0
1354	61.727337	127.0.0.1	127.0.0.1	TCP	61 57501 → 5555 [PSH, ACK] Seq=8 Ack=1 Win=327424 Len=17
1364	62.249497	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=25 Win=327424 Len=0
1365	62.249527	127.0.0.1	127.0.0.1	TCP	58 5555 → 57515 [PSH, ACK] Seq=1 Ack=8 Win=327424 Len=14
1374	62.775488	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=8 Ack=15 Win=327424 Len=0
1481	65.950193	127.0.0.1	127.0.0.1	TCP	61 57501 → 5555 [PSH, ACK] Seq=25 Ack=1 Win=327424 Len=17
1501	66.471329	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=42 Win=327424 Len=0
1502	66.471346	127.0.0.1	127.0.0.1	TCP	58 5555 → 57515 [PSH, ACK] Seq=15 Ack=8 Win=327424 Len=14
1518	66.980445	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=8 Ack=29 Win=327424 Len=0
1575	69.759998	127.0.0.1	127.0.0.1	TCP	61 57501 → 5555 [PSH, ACK] Seq=42 Ack=1 Win=327424 Len=17
1588	70.287101	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=59 Win=327168 Len=0
1589	70.287123	127.0.0.1	127.0.0.1	TCP	58 5555 → 57515 [PSH, ACK] Seq=29 Ack=8 Win=327424 Len=14
1605	70.818226	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=8 Ack=43 Win=327424 Len=0
1642	72.931704	127.0.0.1	127.0.0.1	TCP	61 57501 → 5555 [PSH, ACK] Seq=59 Ack=1 Win=327424 Len=17
1651	73.459828	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=76 Win=327168 Len=0
1652	73.459857	127.0.0.1	127.0.0.1	TCP	58 5555 → 57515 [PSH, ACK] Seq=43 Ack=8 Win=327424 Len=14
1662	73.987940	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=8 Ack=57 Win=327168 Len=0
1717	76.158520	127.0.0.1	127.0.0.1	TCP	61 57501 → 5555 [PSH, ACK] Seq=76 Ack=1 Win=327424 Len=17
1733	76.671643	127.0.0.1	127.0.0.1	TCP	44 5555 → 57501 [ACK] Seq=1 Ack=93 Win=327168 Len=0
1734	76.671661	127.0.0.1	127.0.0.1	TCP	58 5555 → 57515 [PSH, ACK] Seq=57 Ack=8 Win=327424 Len=14
1750	77.202774	127.0.0.1	127.0.0.1	TCP	44 57515 → 5555 [ACK] Seq=8 Ack=71 Win=327168 Len=0

## Wireshark da aplicação TCP com latência variável



udp.port==5555						
No.	Time	Source	Destination	Protocol	Length	Info
326	12.024293	127.0.0.1	127.0.0.1	UDP	39	58559 → 5555 Len=7
544	22.286655	127.0.0.1	127.0.0.1	UDP	39	57907 → 5555 Len=7
918	40.861024	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
928	41.390210	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
946	42.486453	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
961	43.017563	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
981	43.831731	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
988	44.362801	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
1008	44.932043	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
1025	45.463085	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
1040	46.197243	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
1047	46.724363	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
1055	47.416758	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
1067	47.944875	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
1098	48.716751	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
1115	49.245776	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17
1142	50.428620	127.0.0.1	127.0.0.1	UDP	49	58559 → 5555 Len=17
1166	50.957749	127.0.0.1	127.0.0.1	UDP	49	5555 → 57907 Len=17

**Wireshark da aplicação UDP com latência variável**