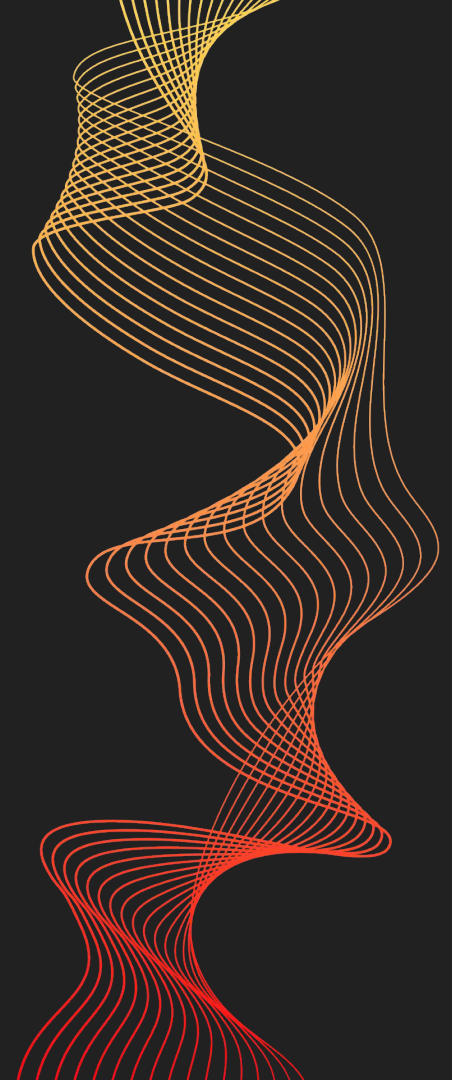




# Redes de Computadores - EP1

Aluna: Laís Nuto Rossman

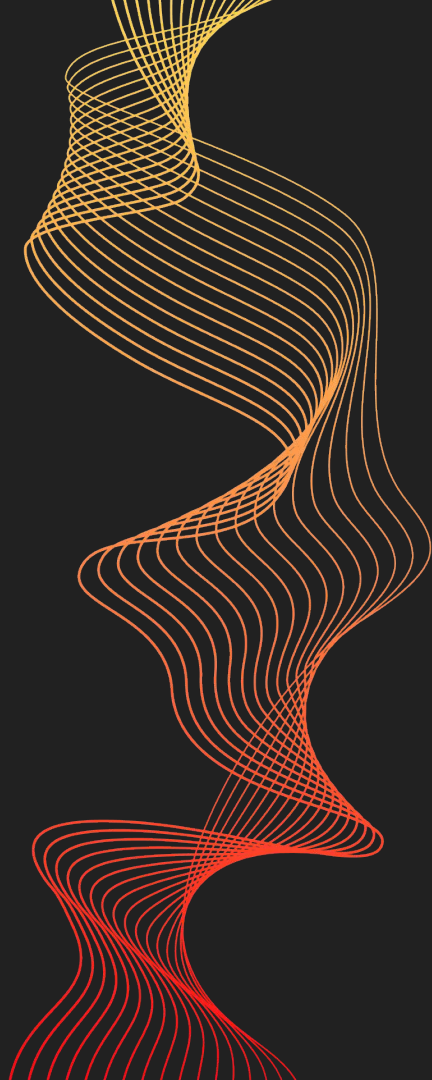
NºUSP: 12547274





# Implementando o AMQP Server

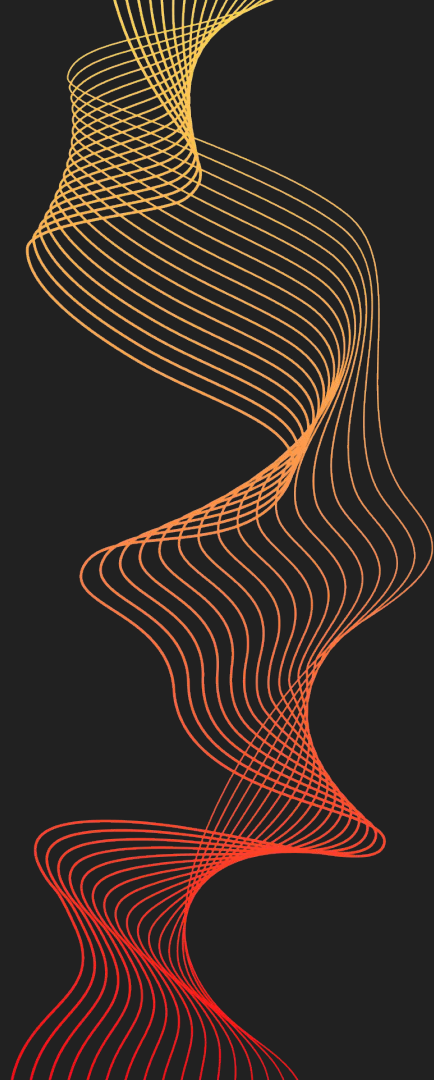
Nessa apresentação vamos ter uma visão geral da implementação de um servidor AMQP na versão 0.9.1 do protocolo usando a linguagem C. O objetivo é buscar entender como o servidor foi feito e realizar alguns testes com vários clientes simultaneamente para verificar seu desempenho





# Implementação: Handshaking

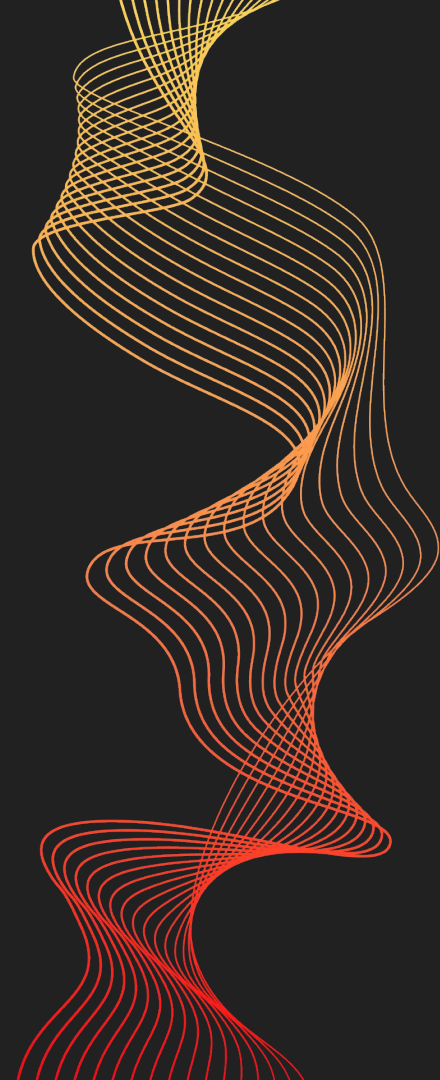
- A função de lidar com o cliente começa estabelecendo todo o processo de handshaking
- O programa lê e escreve os pacotes necessários para a negociação da versão do protocolo, configuração da conexão, abertura da conexão e abertura do canal.
- Após o handshaking, o programa vai ler mais um pacote que vai identificar o que o cliente quer fazer (declare, consume ou publish)





# Implementação: Declare Queue

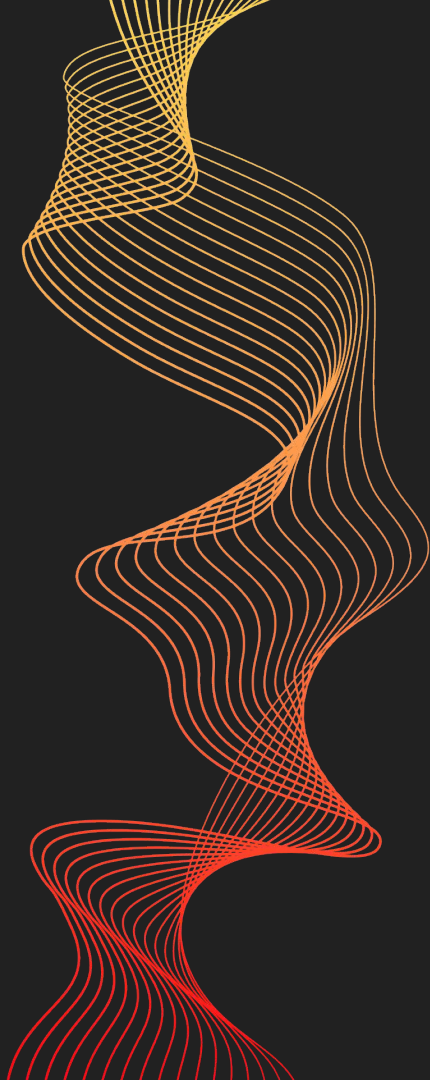
- Se o pacote lido for um Declare Queue, o programa cria uma estrutura de dados que armazena uma fila de mensagens e uma fila circular de consumidores com o mesmo nome da fila criada
- Além de escrever o pacote de declare-queue-ok, o programa escreve e lê os pacotes para fechar o canal e fechar conexão





# Implementação: Publish Queue

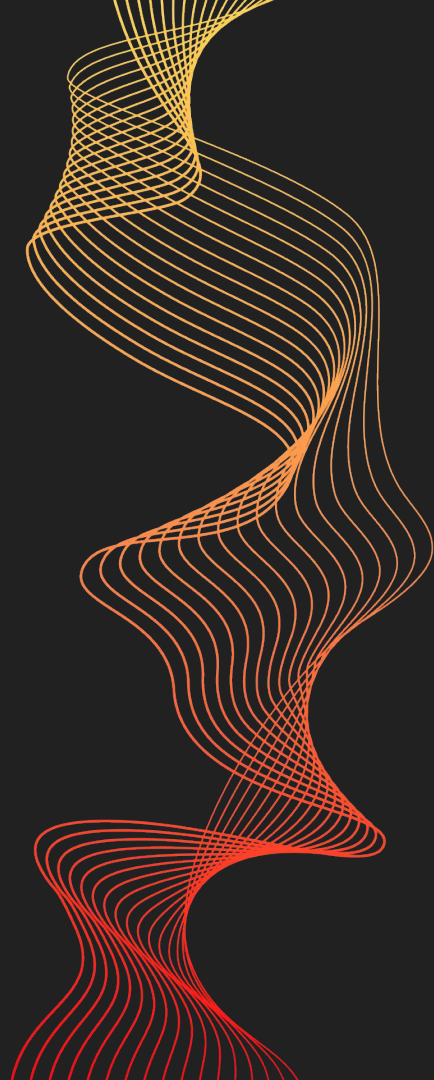
- Se o pacote lido for um publish queue, logo em seguida o programa lê os pacotes de header e body desse publish. Depois disso, o programa escreve e lê os pacotes para fechar o canal e fechar conexão
- Além disso, o programa verifica se já há consumidores para a fila que foi publicada. Se tiver consumidores, pego o consumidor do topo da fila circular e além de escrever os pacotes de consume-ok, o programa faz o processo do basic deliver e depois lê o basic ack
- Vale destacar que depois de consumir a mensagem, o consumidor vai para o final da fila para cumprir o esquema Round Robin





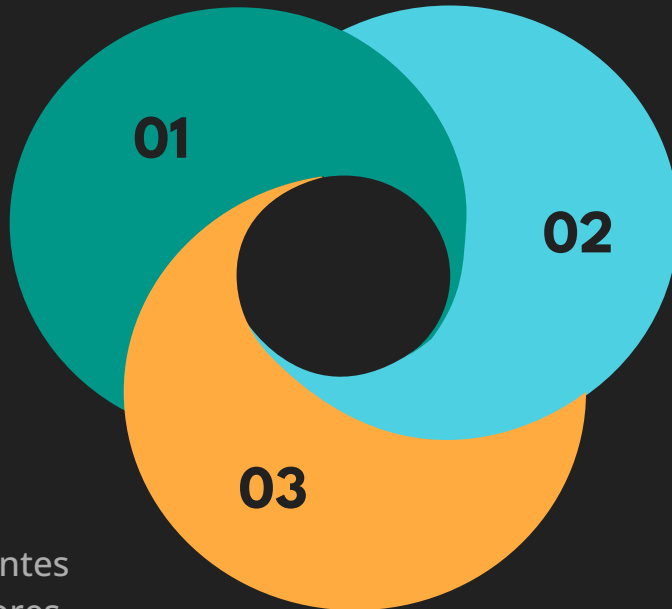
# Implementação: Consume Queue

- Se o pacote lido for um Consume Queue, então colocamos o arquivo descritor (connfd) numa fila circular para identificar aquele consumidor.
- Depois disso, é verificado se o consumidor que foi adicionado é o primeiro da fila e se tem mensagens para ele consumir. Se sim, ele consome as mensagens imediatamente, e o processo do basic deliver é feito e depois o programa lê o basic ack
- Vale destacar que após esses processos, o programa faz um looping infinito que verifica se aquele cliente está conectado. A partir do momento que o cliente interromper o servidor com cntrl c, a conexão é encerrada imediatamente



# Testes do servidor

10 testes com servidor  
sem nenhum cliente  
conectado



10 testes com servidor  
com 10 clientes (metade  
consumidores, outra  
metade publicando nas  
filas)

Servidor com 100 clientes  
(metade consumidores,  
outra metade publicando  
nas filas)

```

      .-/+00SSSS00+/-.,
      `:+SSSSSSSSSSSSSSSS+!`
      -+SSSSSSSSSSSSSSSSyySSSS+-
      .0SSSSSSSSSSSSSSSSdMMMMNySSSS0.
      /SSSSSSSSSShdmmNNmmyNNMMMMhSSSSS/
      +SSSSSSSShmydMMMMMMNdddySSSSSSS+
      /SSSSSSSShNMMMyhhyyyhmmMMMNhSSSSSSS/
      .SSSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
      +SSShhhyNMMNySSSSSSSSSSyNMMMySSSSSSS+
      0SSyNMMMNyMMhSSSSSSSSSSShmmhSSSSSSS0
      0SSyNMMMNyMMhSSSSSSSSSSShmmhSSSSSSS0
      +SSShhhyNMMNySSSSSSSSSSyNMMMySSSSSSS+
      .SSSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
      /SSSSSSSShNMMMyhhyyyhdmMMMNhSSSSSSS/
      +SSSSSSSSdmydMMMMMMNdddySSSSSSS+
      /SSSSSSSSShdmmNNmmyNNMMMMhSSSSS/
      .0SSSSSSSSSSSSSSSSdMMMMNySSSS0.
      -+SSSSSSSSSSSSSSSSyySSSS+-
      `:+SSSSSSSSSSSSSSSS+!`
      .-/+00SSSS00+/-.,

```

laisnuto@laisnuto-550XDA

```

-----
OS: Ubuntu 20.04.6 LTS x86_64
Host: 550XDA P17CFB
Kernel: 5.15.0-87-generic
Uptime: 20 hours, 5 mins
Packages: 2082 (dpkg), 11 (snap)
Shell: bash 5.0.17
Resolution: 1920x1080
DE: GNOME
WM: Mutter
WM Theme: Adwaita
Theme: Yaru [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: gnome-terminal
CPU: 11th Gen Intel i5-1135G7 (8) @
GPU: Intel Device 9a49
Memory: 11082MiB / 15712MiB

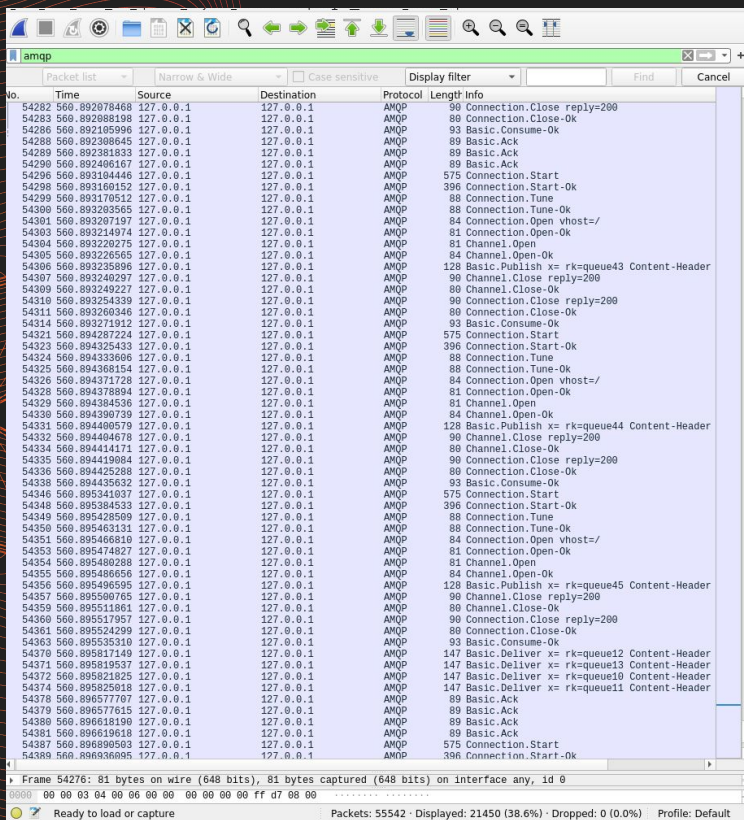
```



## Informações de ambiente de teste

- Processador: i5-1135G7 2.40GHz x86\_64
- Versão do gcc: gcc 9.4.0
- Sistema Operacional: Ubuntu 20.04.4 LTS
- Shell: bash 5.0.17
- Bibliotecas:
  - `pthread` para suporte a multithreading.





No.	Time	Source	Destination	Protocol	Length	Info
54282	560.892078468	127.0.0.1	127.0.0.1	AMQP	90	Connection.Close reply=200
54283	560.892088198	127.0.0.1	127.0.0.1	AMQP	80	Connection.Close-Ok
54286	560.892105999	127.0.0.1	127.0.0.1	AMQP	93	Basic.Consume-Ok
54288	560.892388645	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54289	560.892381833	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54290	560.892406167	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54296	560.893104446	127.0.0.1	127.0.0.1	AMQP	575	Connection.Start
54298	560.893160152	127.0.0.1	127.0.0.1	AMQP	396	Connection.Start-Ok
54299	560.893170512	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune
54300	560.893205565	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune-Ok
54301	560.893207197	127.0.0.1	127.0.0.1	AMQP	84	Connection.Open vhost=/
54303	560.893214974	127.0.0.1	127.0.0.1	AMQP	81	Connection.Open-Ok
54304	560.893226275	127.0.0.1	127.0.0.1	AMQP	81	Channel.Open
54305	560.893226565	127.0.0.1	127.0.0.1	AMQP	84	Channel.Open-Ok
54306	560.893235896	127.0.0.1	127.0.0.1	AMQP	128	Basic.Publish x= rk=queue43 Content-Header
54307	560.893240297	127.0.0.1	127.0.0.1	AMQP	90	Channel.Close reply=200
54309	560.893249227	127.0.0.1	127.0.0.1	AMQP	80	Channel.Close-Ok
54310	560.893254339	127.0.0.1	127.0.0.1	AMQP	90	Connection.Close reply=200
54311	560.893260346	127.0.0.1	127.0.0.1	AMQP	80	Connection.Close-Ok
54314	560.893271912	127.0.0.1	127.0.0.1	AMQP	93	Basic.Consume-Ok
54321	560.894287224	127.0.0.1	127.0.0.1	AMQP	575	Connection.Start
54323	560.894325433	127.0.0.1	127.0.0.1	AMQP	396	Connection.Start-Ok
54324	560.894333606	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune
54325	560.894368154	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune-Ok
54326	560.894371728	127.0.0.1	127.0.0.1	AMQP	84	Connection.Open vhost=/
54328	560.894378894	127.0.0.1	127.0.0.1	AMQP	81	Connection.Open-Ok
54329	560.894384536	127.0.0.1	127.0.0.1	AMQP	81	Channel.Open
54330	560.894390729	127.0.0.1	127.0.0.1	AMQP	84	Channel.Open-Ok
54331	560.894400579	127.0.0.1	127.0.0.1	AMQP	128	Basic.Publish x= rk=queue44 Content-Header
54332	560.894404678	127.0.0.1	127.0.0.1	AMQP	90	Channel.Close reply=200
54334	560.894414171	127.0.0.1	127.0.0.1	AMQP	80	Channel.Close-Ok
54335	560.894419984	127.0.0.1	127.0.0.1	AMQP	90	Connection.Close reply=200
54336	560.894425288	127.0.0.1	127.0.0.1	AMQP	80	Connection.Close-Ok
54338	560.894435632	127.0.0.1	127.0.0.1	AMQP	93	Basic.Consume-Ok
54340	560.895241937	127.0.0.1	127.0.0.1	AMQP	575	Connection.Start
54348	560.895384533	127.0.0.1	127.0.0.1	AMQP	396	Connection.Start-Ok
54349	560.895428509	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune
54350	560.895463131	127.0.0.1	127.0.0.1	AMQP	88	Connection.Tune-Ok
54351	560.895466910	127.0.0.1	127.0.0.1	AMQP	84	Connection.Open vhost=/
54353	560.895474827	127.0.0.1	127.0.0.1	AMQP	81	Connection.Open-Ok
54354	560.895480288	127.0.0.1	127.0.0.1	AMQP	81	Channel.Open
54355	560.895480656	127.0.0.1	127.0.0.1	AMQP	84	Channel.Open-Ok
54356	560.895496595	127.0.0.1	127.0.0.1	AMQP	128	Basic.Publish x= rk=queue45 Content-Header
54357	560.895500765	127.0.0.1	127.0.0.1	AMQP	90	Channel.Close reply=200
54359	560.895511861	127.0.0.1	127.0.0.1	AMQP	80	Channel.Close-Ok
54360	560.895517957	127.0.0.1	127.0.0.1	AMQP	90	Connection.Close reply=200
54361	560.895524299	127.0.0.1	127.0.0.1	AMQP	80	Connection.Close-Ok
54363	560.895535310	127.0.0.1	127.0.0.1	AMQP	93	Basic.Consume-Ok
54370	560.895817140	127.0.0.1	127.0.0.1	AMQP	147	Basic.Deliver x= rk=queue12 Content-Header
54371	560.895819537	127.0.0.1	127.0.0.1	AMQP	147	Basic.Deliver x= rk=queue13 Content-Header
54372	560.895821825	127.0.0.1	127.0.0.1	AMQP	147	Basic.Deliver x= rk=queue10 Content-Header
54374	560.895825018	127.0.0.1	127.0.0.1	AMQP	147	Basic.Deliver x= rk=queue11 Content-Header
54376	560.896577707	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54379	560.896577615	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54380	560.896618190	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54381	560.896619616	127.0.0.1	127.0.0.1	AMQP	89	Basic.Ack
54387	560.896890563	127.0.0.1	127.0.0.1	AMQP	575	Connection.Start
54389	560.896936095	127.0.0.1	127.0.0.1	AMQP	396	Connection.Start-Ok

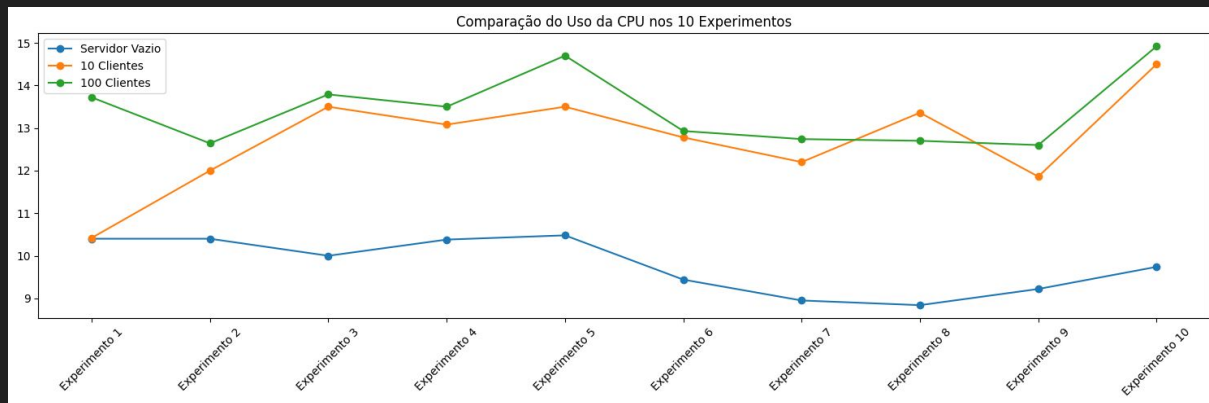
Frame 54276: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface any, id 0  
 0000 00 00 03 04 00 06 00 00 00 00 00 00 ff d7 08 00  
 Ready to load or capture Packets: 55542 - Displayed: 21450 (38.6%) - Dropped: 0 (0.0%) - Profile: Default

## Detalhes dos testes

- Os testes foram feitos a partir de um script que declarava filas equivalentes a metade do número de clientes, publicava e consumia nessas filas
- Enquanto os testes rodavam, o wireshark capturava os pacotes para verificar se o resultado era como esperado (como ao lado)
- Os dados de uso da rede foram capturados usando o tshark e o filtro de amqp
- Os dados da CPU foram calculados a partir da sua ociosidade usando o top
- Além disso, o programa foi testado em outra máquina apenas para verificar que funciona mesmo em cenários onde as conexões não venha do localhost.

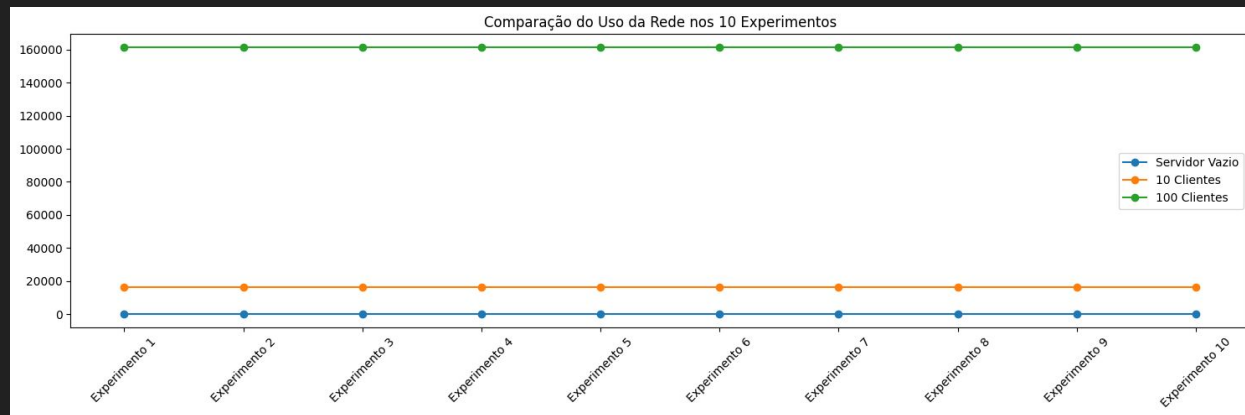
## ▶ Uso da CPU nos 3 casos

- No estado de vazio, a CPU tem um uso menor, variando entre 8.84% e 10.48%. Quando há 10 clientes conectados, o uso da CPU aumenta e fica entre 10.42% e 14.5%. Com número de clientes aumentando para 100 (10 vezes o tráfego anterior), o uso da CPU também aumenta, mas não em uma proporção diretamente proporcional ao aumento da rede, variando entre 12.6% e 14.92%. É importante verificar que o uso da CPU inclui o que foi usado para rodar o script, o que significa que analisar a comparação entre os experimentos traz mais respostas do que ver os valores de uso da CPU em si



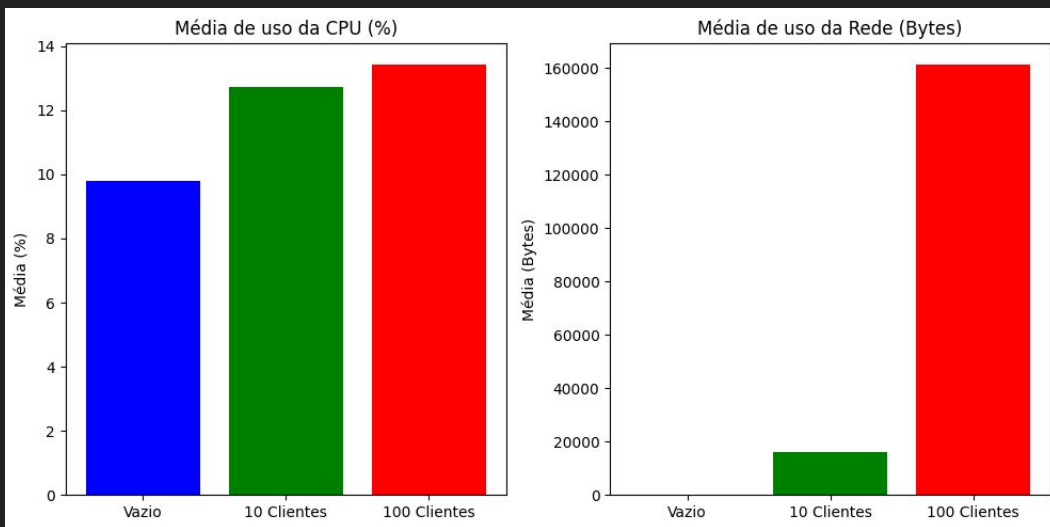
## ► Uso da Rede nos 3 casos

- O gráfico mostra a quantidade total de dados (em bytes) que foram transmitidos/recebidos na interface de loopback (lo) na porta amqp durante a execução do experimento. Em cada experimento, o tráfego de rede permanece constante (0 para nenhum cliente conectado, 16110 para tráfego de 10 e 161305 para tráfego de 100), o que mostra a consistência em relação carga de rede.




## ▶ Análise dos resultados

- O tráfego de rede parece impactar o uso da CPU, pois conforme o tráfego aumenta, o uso da CPU também aumenta.
- No entanto, a relação não parece ser linear. Mesmo quando o tráfego de rede aumenta 10 vezes, o aumento no uso da CPU não é tão significativo.





## Conclusão

- Quando o tráfego de rede aumenta significativamente, e o uso da CPU não acompanha de forma proporcional, mostrando a eficiência do sistema com um aumento no tráfego sem exigir muito mais recursos de CPU.
  - Assim, o sistema tem uma boa capacidade de ser escalável vendo a forma com que ele lida com o tráfego de rede.
- 



Obrigado pelo seu tempo e atenção 😊