EP2 - MAC0352

Caio Fontes de Castro - 10692061
Leandro Rodrigues - 10723944

Implementação

Detalhes gerais

- O trabalho foi realizado na linguagem *python,* utilizando apenas bibliotecas padrões da linguagem.
  - Destacamos os módulos *socket,* e *ssl*, utilizados para a comunicação cliente-servidor e cliente-cliente.
- As mensagens, através de um protocolo desenvolvido por nós mesmos, trocadas são estruturadas como blocos de texto, e utilizamos o caractere “;” para separar os blocos. Por exemplo, o comando *begin:*
  - O cliente digita no seu prompt:

```
>begin otheruser
```

Isso seria traduzido para a mensagem:

```
“begin;otheruser”
```

Conexões

- Cada Cliente cria três conexões com o servidor, através de duas sockets diferentes.
  - Uma conexão TLS, para os comandos *login, adduser* e *passwd*
  - Uma conexão normal, não encriptada, para os outros comandos.
  - Uma conexão para receber *heartbeats* do servidor.
  - A conexão não encriptada se dá pela mesma porta para todos os clientes (porta 3000 por padrão), que deve ser informada quando o programa *start_client* é iniciado.
  - A partir dessa conexão o cliente recebe a porta pela qual deve realizar a conexão TLS do servidor. (se não estiver ocupada, 3001).
- A conexão p2p entre os dois clientes em um jogo também se dá por dois sockets, os dois não encriptados.
  - Um socket para troca de jogadas em si.
  - Um socket para medição do *delay*.

Servidor

- O servidor possui uma implementação baseada em threads.
  - A thread principal fica sempre a escuta de novas conexões na porta principal.
  - Quando uma conexão nova é criada, uma nova thread também é criada. Essa thread é a encarregada de lidar com os requests feitos através desta conexão.
- A comunicação entre as threads se dá através de dicionários indexados pelos ids únicos de cada thread.
- Os arquivos *users.txt*, *log.txt*, *cert.pem* e *key.pem* são arquivos auxiliares utilizados pelo servidor.

Cliente

- O cliente também utiliza threads, mas seu funcionamento para o usuário é basicamente linear
- Ao executar o script, um prompt é iniciado, e aguarda por comandos do usuário. Exemplo do funcionamento:

```
$ python start_client.py
>leaders
----------------------
     LEADERBOARD
----------------------
  leandro    | 3   |
     caio |    1   |
>list
------------------------------
Usuários conectados:
------------------------------
>login caio a
Login bem sucedido!
>^CFinalizando cliente
$
```

Comandos

list

- O cliente envia um pacote “list” e recebe um pacote “listACK”, com a lista de nomes dos usuário logados.

leaders

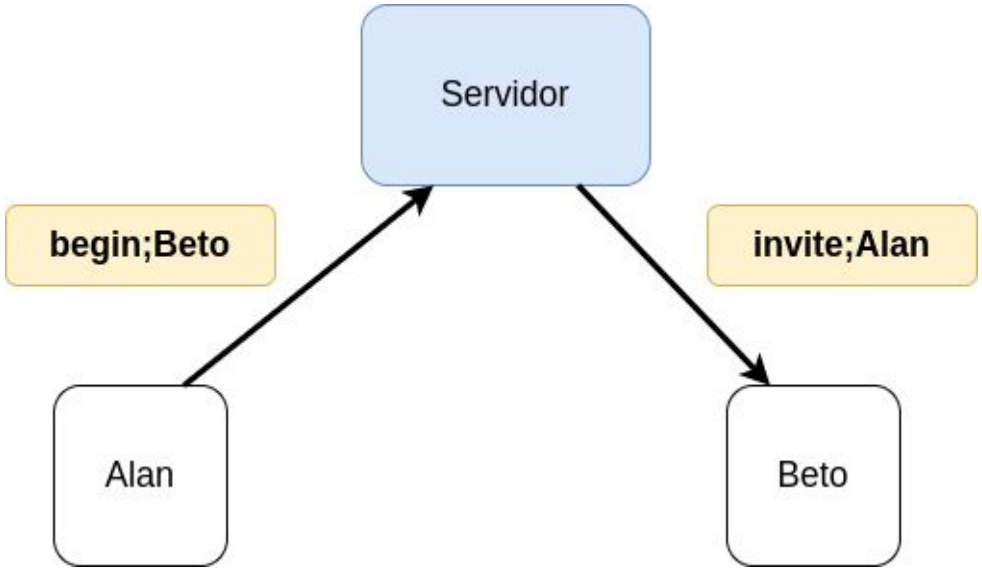
- O cliente envia um pacote “leaders” e recebe um pacote “leadersACK”, com uma lista de tuplas “nome de usuário”:”pontuação”. A lista não é ordenada, deve ser ordenada pelo cliente.

begin

- O comando begin possui várias checagens para garantir que partida pode acontecer, então vamos descrever o fluxo de mensagens trocadas para dois jogadores já logados iniciarem uma partida.
    - Todas as mensagens *begin, invite* e *answer* são respondidas adequadamente com mensagens ...*ACK* se não houver problemas ou ...*ERR;mensagem de erro* caso haja alguma coisa errada.
    - Se houver algum erro, todo o processo de convite é reiniciado.
- Supomos 2 jogadores chamados Alan e Beto que já logaram no servidor e não estão atualmente em uma partida.

begin

1. Primeiramente Alan manda uma mensagem *begin* para o servidor, indicando que quer convidar Beto para uma partida.
2. O servidor verifica que Beto está online, e envia uma mensagem *invite* para Beto com o nome de Alan.



begin

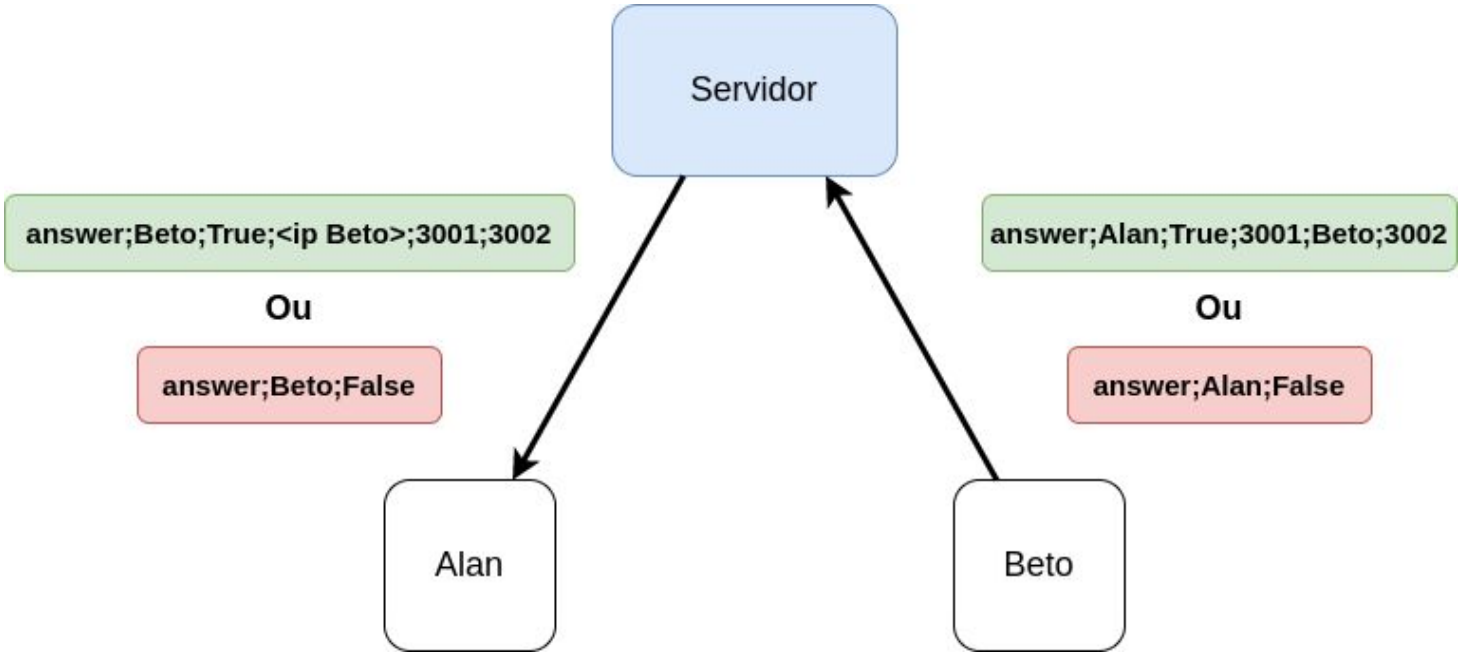
3. Beto então recebe um prompt no seu cliente (após apertar Enter para receber um update):

```
>
>
Alan te convidou para jogar!
Aceitar?
[S/n]:
```

4. Beto então responde com “S” ou “n”, indicando se quer que a partida aconteça.

begin

5. Os pacotes verdes são enviados com uma resposta positiva de Beto, e os vermelhos com uma resposta negativa.



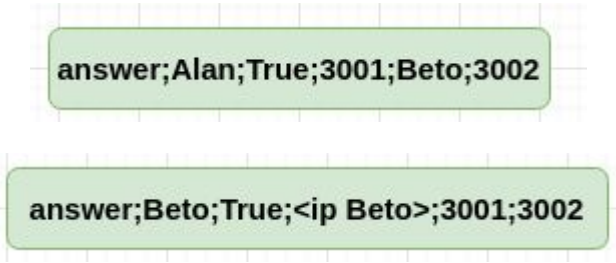
begin

Nos pacotes ao lado, 3001 é a porta que será utilizada para a troca de mensagens do jogo entre os clientes, e 3002 é a porta que será utilizada para a troca de pings.

Esses valores são apenas exemplos e podem ser alterados.

6. Beto e Alan então estabelecem as duas conexões entre si, e começam a partida.

7. Ao final da partida os dois enviam o resultado para o servidor através de uma mensagem *result*



Armazenamento

Usuários

Logins, senhas e placar de usuários são gravados em *plain text,* com os campos separados por um tab em um arquivo users.txt.

Em um cenário ideal, o *hash* criptográfico das senhas seriam guardadas em um arquivo protegido.

Exemplo de arquivo:

```
caio  123456
leandro abcdefg
beto  beto1234
alice alice99999
```

Logs

O servidor registra algumas informações num arquivo log.txt, como inicialização, conexão/desconexão de um cliente, login, inicio de uma partida, etc.

As linhas de log são prefixadas pela data no formato [%Y-%m-%d %H:%M:%S]

Exemplo do arquivo de logs:

```
[2021-06-20 15:50:00] Server started listening in port 3050
[2021-06-20 15:50:03] Client connected from ip 127.0.0.1 and port 58914
[2021-06-20 15:51:29] User 'leandro' successfuly logged in from ip 127.0.0.1
[2021-06-20 15:55:22] Server shutting down
```

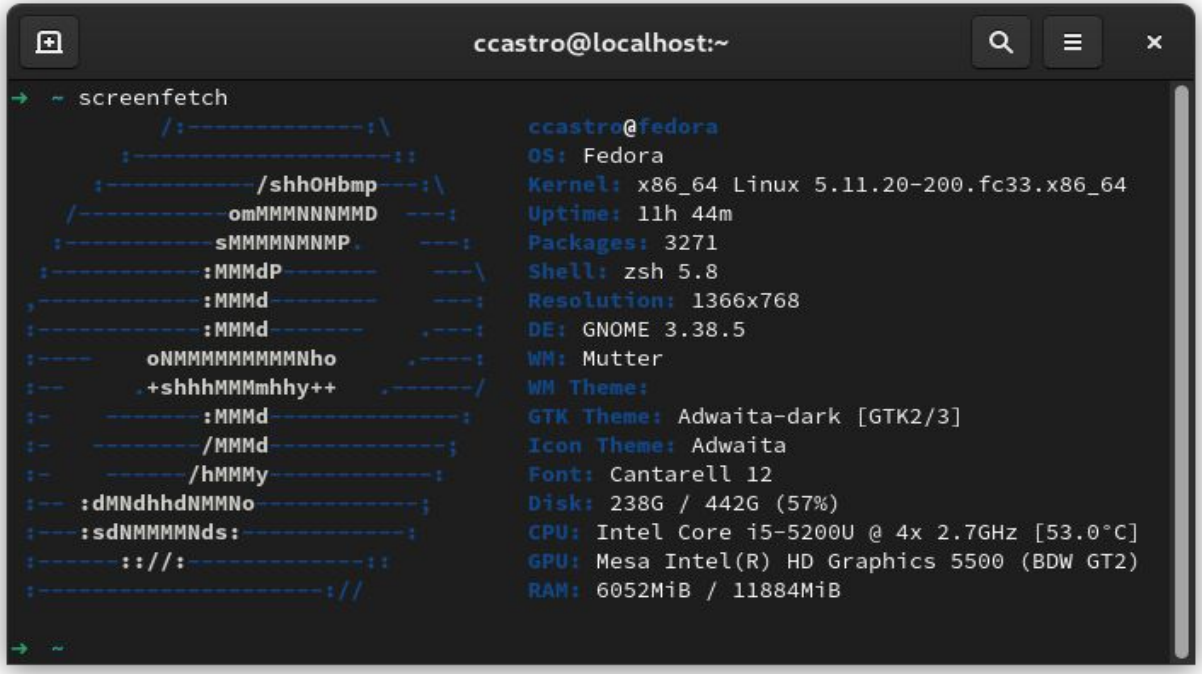
Testes

Informações Sobre a Captura

- As informações sobre utilização de CPU e memória foram capturadas utilizando o programa *psrecord*.
  - O comando específico utilizado para medir o gasto de recursos do servidor foi (exemplo do teste básico):
    - `psrecord "python start_server.py" --log activity_server.txt  --plot plot.png --duration 10 --include-children`
    - `psrecord "python start_client.py" --log activity_client.txt  --plot plot.png --duration 10 --include-children`
  - Esse comando garante a captura apenas do consumo de recursos do próprio servidor.
  - O gráfico gerado mostra o Consumo de CPU (em %) e memória (em Mb).
- As informações de Rede foram coletadas através do programa *tshark*.
  - O comando específico utilizado para medir a utilização de rede foi (exemplo do teste básico):
    - `tshark -i "loopback" -w capture.pcap`
  - A partir dessa captura, geramos os gráficos apresentados, que mostram a quantidade de bytes transmitidos APENAS nos pacotes trocados pelo nosso programa.

Ambiente de teste

- Todos os testes foram executados em uma rede Wifi local. Tanto o servidor quanto os clientes foram executados em um Laptop Lenovo G40-80, com os detalhes abaixo:



Descrição dos Testes

Teste Servidor	Apenas o servidor, sem clientes.
Teste Básico	Servidor, 2 Clientes logados, sem partidas.
Teste Partida	Servidor, 2 Clientes logados, em uma partida.

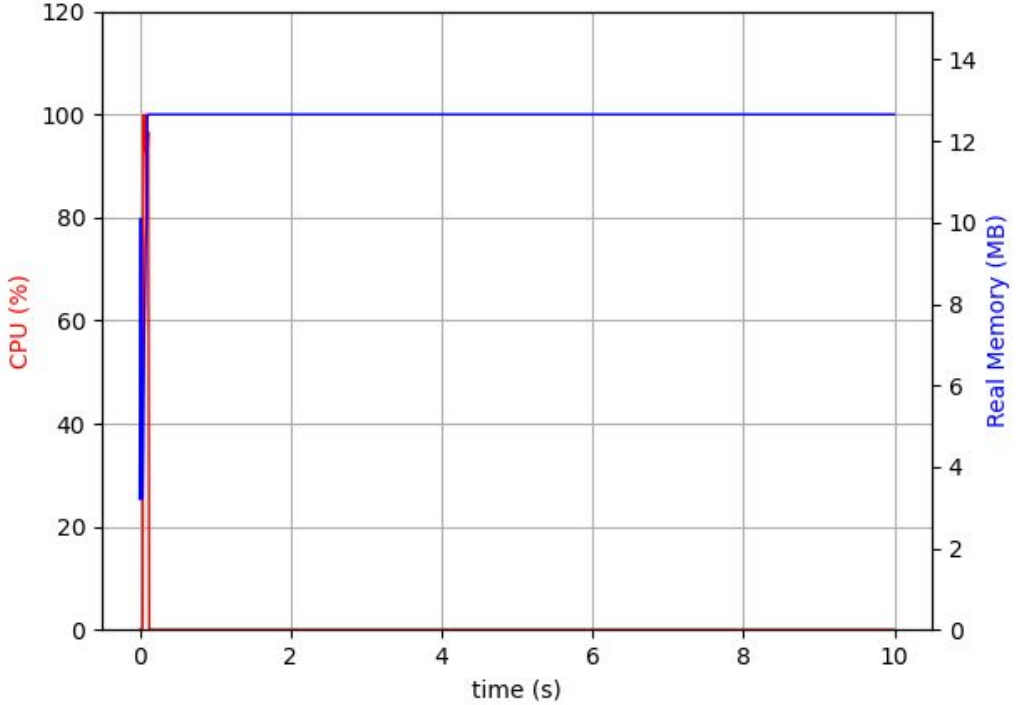
Teste servidor

Apenas o servidor, sem clientes conectados.

O servidor fica no ar por 10 segundos e é desligado.

Temos um consumo de CPU alto no início enquanto o server se preprara, mas quando o servidor entra em espera por conexões o consumo é quase nulo.

A memória alocada tem um crescimento rápido no inicio e também se estabiliza.

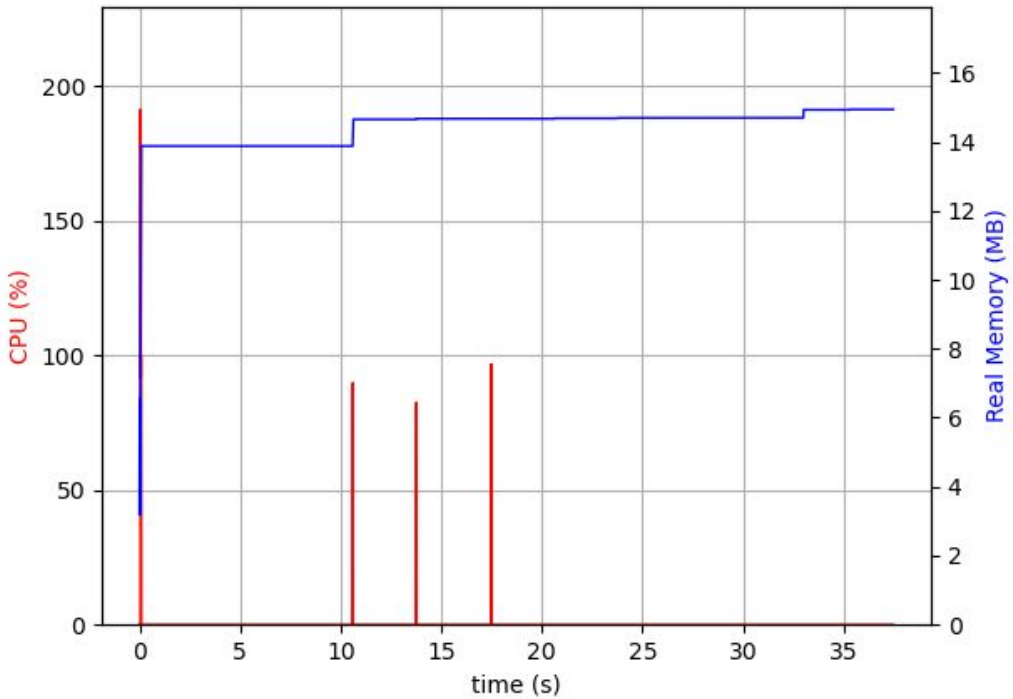


Teste básico

Servidor, mais 2 clientes logados.

Ao lado temos a performance do servidor.

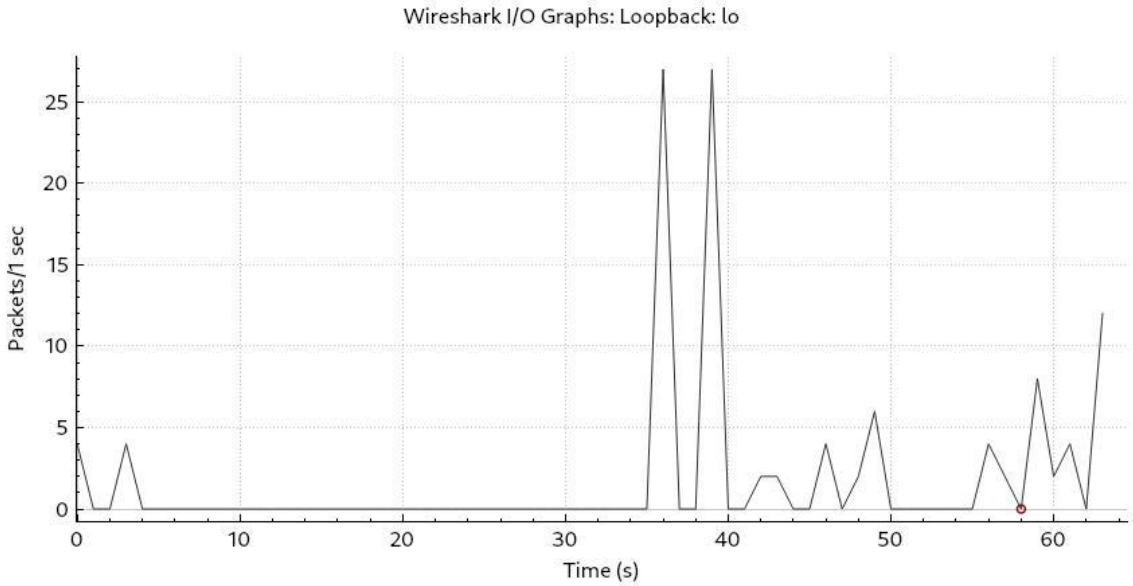
Podemos ver o mesmo pico no setup inicial, tanto no consumo de CPU quanto de memória, com pequenos picos na troca de mensagens com os clientes.



Teste básico

Servidor, mais 2 clientes logados.

Podemos ver os picos no consumo de rede nas conexões iniciais e no login dos usuários, e depois um consumo menor devido ao heartbeat.

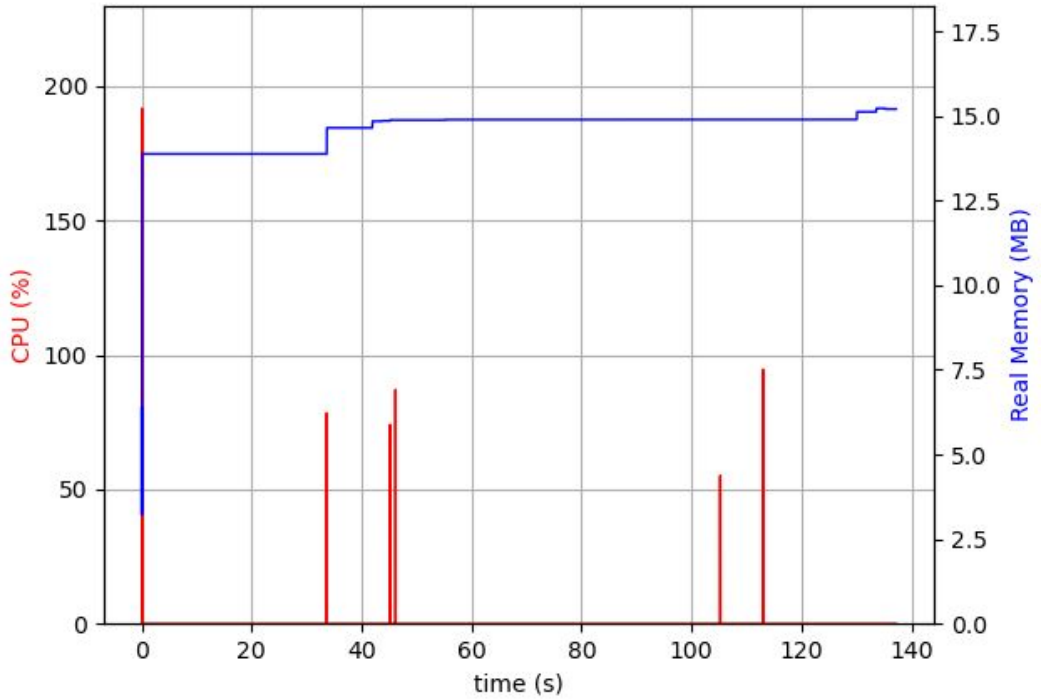


Teste Partida

Servidor, mais 2 clientes logados em uma partida.

Ao lado temos a performance do servidor.

Podemos ver o mesmo pico no setup inicial, tanto no consumo de CPU quanto de memória, com pequenos picos na troca de mensagens com os clientes. Os últimos picos são causados pelos clientes reportando os resultados.

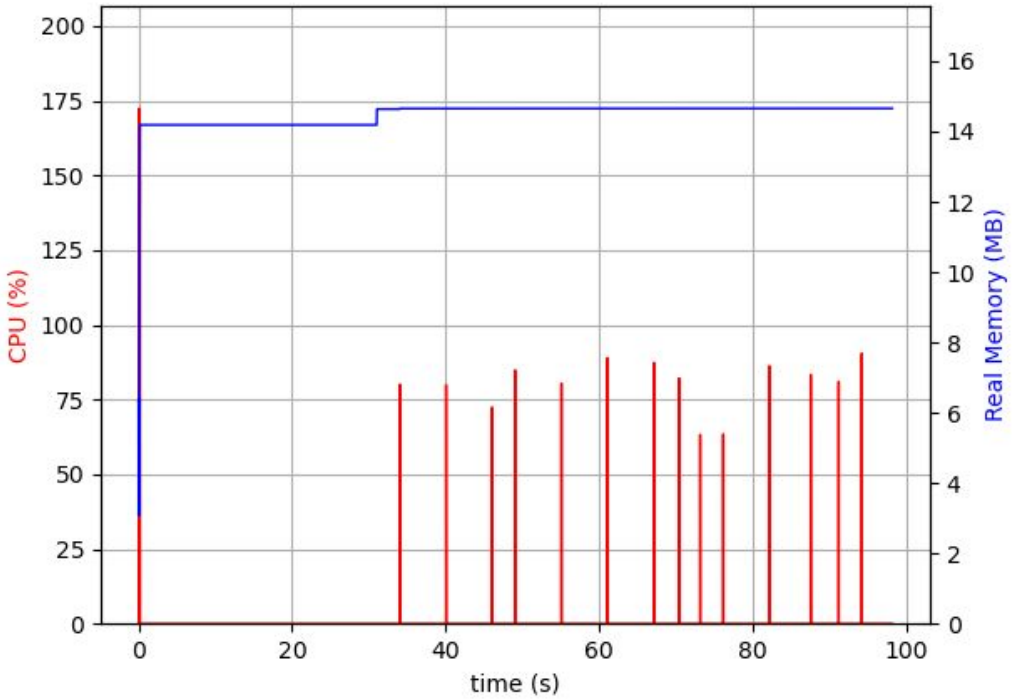


Teste Partida

Servidor, mais 2 clientes logados em uma partida.

Ao lado temos a performance do cliente.

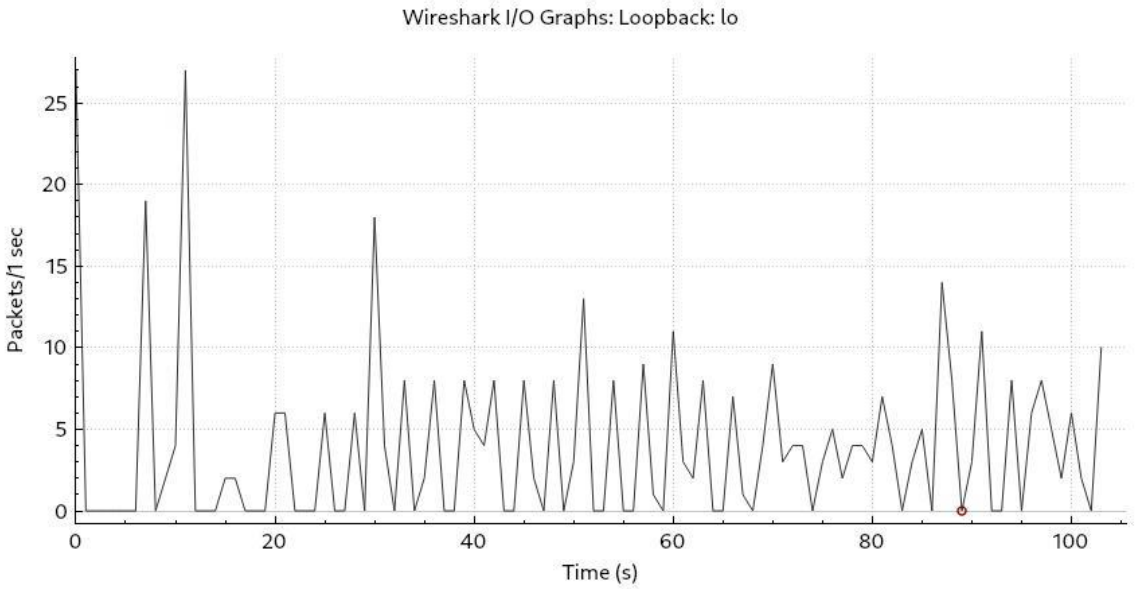
Podemos ver o mesmo padrão de picos durante a troca de mensagens pelas jogadas.



Teste Partida

Servidor, mais 2 clientes logados em uma partida.

Ao lado temos o consumo de rede durante a partida, tanto da troca de mensagens entre os clientes quanto com o servidor.



Conclusões

Conclusões

- O servidor e cliente implementados apresentam as funcionalidades pedidas, e podem ser utilizados da maneira proposta para o jogo.
- Existem limitações de performance e consumo de rede aparentes, que deveriam ser evitadas em um servidor em produção.
- A recuperação de falhas foi apenas parcialmente implementada, não atingindo todos os pontos pedidos no enunciado.
    - Por exemplo, se o servidor cair e voltar a funcionar os clientes não conseguem se reconectar, e devem ser reiniciados.