ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos
5º semestre de 2024.

EACH/USP

EXERCÍCIO PROGRAMA

Ana Clara das Neves Barreto - 13672540
Marcos Martins de Oliveira - 13672602

PERÍODO NOTURNO, Turma 04

Disciplina: **Desenvolvimento de Sistemas de Informação Distribuídos**
Prof. Dr. Renan Cerqueira Afonso Alves

5º semestre/2024

Resumo

Esse documento visa explicar e exemplificar o exercício programa sobre um sistema peer to peer não estruturado que realiza as determinadas buscas: inundação (flooding), caminhada aleatória (random walk) e busca em profundidade. E será separado em três partes:

1. Detalhes de implementação
2. Testes de funcionamento
3. Análise de dados

0.1 Detalhes de implementação

Essa sessão irá detalhar mais precisamente algumas informações relacionadas a organização do código e princípios utilizados para sua construção.

Optamos por separar o trabalho em algumas pastas, sendo elas:

- Grafo: contém o arquivo buscas.py (que implementa diferentes algoritmos de busca, sendo eles flooding, random walk, busca em profundidade. Ele usa a instância de 'Peer' passada no constutor para interagir com outros peers)
- Interface: contém o arquivo programa.py (contém a inicialização do programa com a função main, instancia Peer e Buscas e coordena suas operações com base nas escolhas do usuário)
- outros arquivos soltos, como peer.py (responsável por gerenciar a comunicação entre peers, vizinhos, e o armazenamento de dados, inicialização do peer e outras funções de conexão, como iniciar servidor, enviar mensagens, etc)

A inicialização deve ocorrer no arquivo programa.py e é importante que todos os arquivos (de vizinhos e de chaves valor) estejam na mesmo diretório interface, ou adaptar a entrada para que eles consigam ser acessados.

Optamos por usar python e orientação a objetos, definindo as classes 'Peer', 'Buscas' e 'Interface', que são os principais blocos de construção, com cada classe encapsulando determinados dados e métodos. Instâncias dessas classes são criadas e usadas como objetos a partir daí, como 'self.peer' e 'self.buscas'.

O código foi separado nas seguintes threads:

- Inicialização do servidor em Peer.___init___ para garantir que os servidores possam aceitar conexões de forma separada e o código continua executando.
- Para a chamada do método 'handle_peer', quando uma conexão é aceita por servidor.accept( ), uma nova thread é criada para lidar com essa conexão específica. A própria função 'handle_peer', portanto, lida com cada conexão de uma forma separada, recebendo a mensagem do peer, processando-a e enviando uma resposta apropriada.

Ao longo do código, foram usadas operações bloqueantes em relação à comunicação de rede e manipulação de arquivos, como por exemplo nos métodos:

- conecta_peer( ): faz o uso de sockets. Quando um socket é criado e uma conexão é estabelecida, o programa aguarda até que a conexão seja concluída ou falhe.
- envia_mensagem( ): envia dados pela rede e espera até que a mensagem seja enviada completamente

Na parte de manipulação dos arquivos, também podemos considerar as funções como bloqueantes. Enquanto o arquivo está sendo lido, o programa espera até que a leitura seja concluída.

0.2 Testes de funcionamento

Agora, vamos mostrar alguns testes sobre algumas funções requeridas, como funcionamento correto dos métodos das operações de SEARCH, HELLO e BYE, por exemplo.

0.2.1 Funcionamento das operações HELLO e BYE

A operação HELLO é usada de duas maneiras diferentes:

- na adição de novos vizinhos a partir do txt passado em prompt
- através do comando "HELLO"presente no menu

As Figuras 1, 2 e 3 mostram um exemplo do uso da operação HELLO para a primeira situação, usando a topologia "ciclo 3".

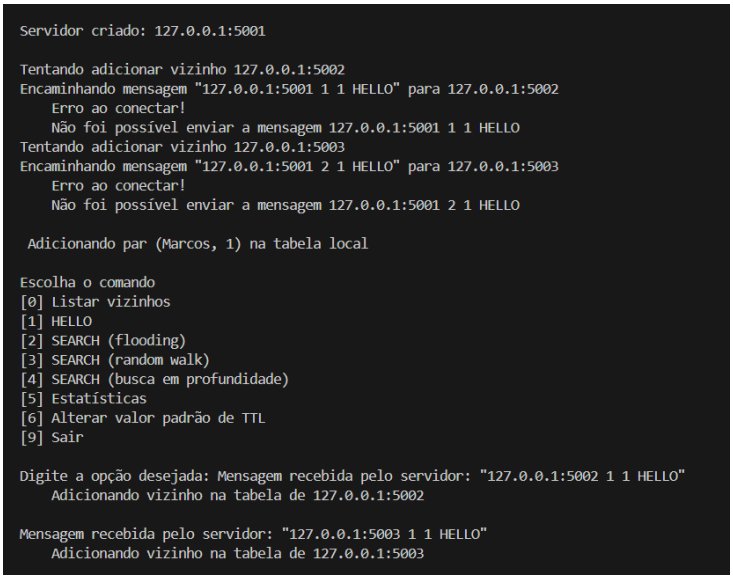

Figura 1 – Inicialização do peer com porta 5001 - Topologia Ciclo 3

```
Servidor criado: 127.0.0.1:5002

Tentando adicionar vizinho 127.0.0.1:5001
Encaminhando mensagem "127.0.0.1:5002 1 1 HELLO" para 127.0.0.1:5001
    Envio feito com sucesso: 127.0.0.1:5002 1 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5003
Encaminhando mensagem "127.0.0.1:5002 2 1 HELLO" para 127.0.0.1:5003
    Erro ao conectar!
    Não foi possível enviar a mensagem 127.0.0.1:5002 2 1 HELLO

 Adicionando par (Gabigol, 1) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: Mensagem recebida pelo servidor: "127.0.0.1:5003 2 1 HELLO"
    Adicionando vizinho na tabela de 127.0.0.1:5003
```

Figura 2 – Inicialização do peer com porta 5002 - Topologia Ciclo 3

```
Servidor criado: 127.0.0.1:5003

Tentando adicionar vizinho 127.0.0.1:5001
Encaminhando mensagem "127.0.0.1:5003 1 1 HELLO" para 127.0.0.1:5001
    Envio feito com sucesso: 127.0.0.1:5003 1 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5002
Encaminhando mensagem "127.0.0.1:5003 2 1 HELLO" para 127.0.0.1:5002
    Envio feito com sucesso: 127.0.0.1:5003 2 1 HELLO

 Adicionando par (Clara, 1) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada:
```

Figura 3 – Inicialização do peer com porta 5003 - Topologia Ciclo 3

Já as Figura 4 e 5, mostram um exemplo do uso da mesma operação para a segunda situação, também utilizando a topologia "ciclo 3". Note que nesse caso, nenhum vizinho é adicionado, pois está enviando HELLO para um vizinho já presente na lista de vizinhos.

```
Digite a opção desejada: 1
Há 2 vizinhos na tabela:
[0] 127.0.0.1:5001
[1] 127.0.0.1:5002
Escolha o vizinho para enviar HELLO: 1
Encaminhando mensagem "127.0.0.1:5003 3 1 HELLO" para 127.0.0.1:5002
    Envio feito com sucesso: 127.0.0.1:5003 3 1 HELLO
```

Figura 4 – SEND HELLO peer 5003 para 5002 - Topologia Ciclo 3

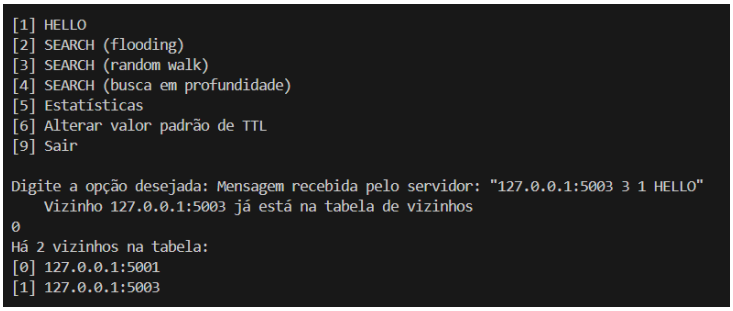

Figura 5 – Peer 5002 recebendo o HELLO do peer 5003 - Topologia Ciclo 3

A operação BYE é usada quando o usuário digita a opção 9 no MENU e ele retira todos os vizinhos da lista, enviando uma mensagem para que ele seja retirado da lista deles também, e encerra o programa. As Figuras 6, 7 e 8 exemplificam essa situação também usando a tipologia "ciclo 3".

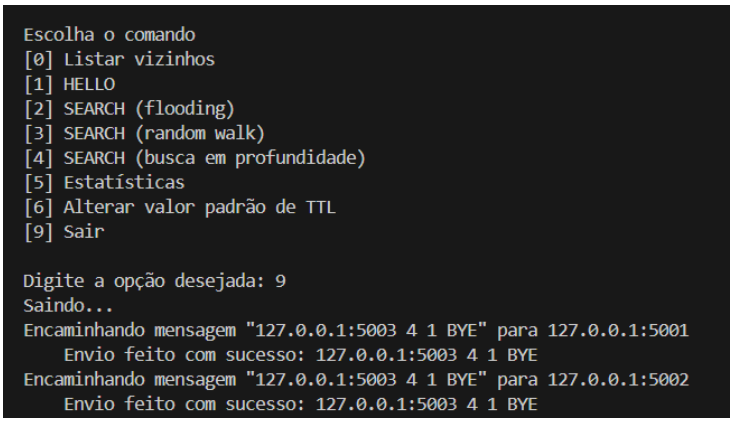

Figura 6 – Enviando BYE com peer 5003 - Topologia Ciclo 3

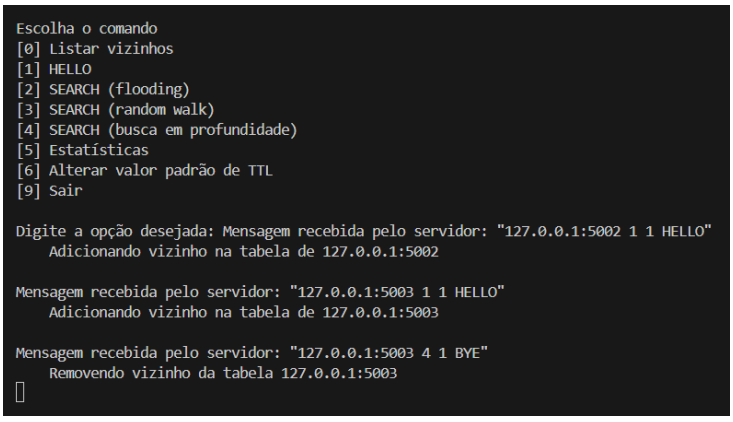

Figura 7 – Peer 5001 recebendo BYE do peer 5003 - Topologia Ciclo 3

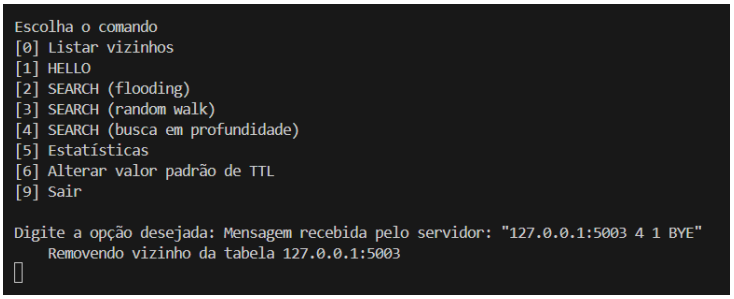

Figura 8 – Peer 5002 recebendo BYE do peer 5003 - Topologia Ciclo 3

0.2.2 Lógica do TTL

O TLL é uma variável fundamental para o processo usado principalmente, nas buscas, dando um limite máximo (estabelecido préviamente como 100), mas que pode também ser alterado no MENU.

A Figura 9 mostra o TTL sendo alterado no menu para 0 (para que possamos ver o TTL sendo eficientemente alterado pelo comando 6) e as Figuras 10, 11 e 12 mostram o que acontece em cada tipo de busca caso o TTL chegue a 0.

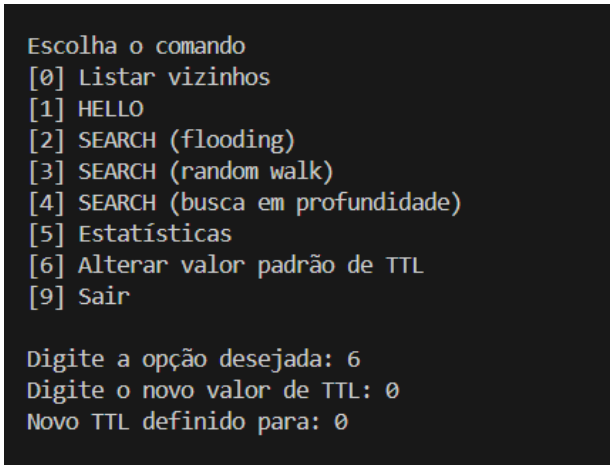

Figura 9 – TLL sendo alterado para 0 - Topologia Ciclo 3

```
Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: 2
Digite a chave a ser buscada: Marcos
 TTL igual a zero, descartando mensagem
Chave não encontrada
```

Figura 10 – TLL chegando a 0 na busca por 'flooding' - Topologia Ciclo 3

```
Digite a opção desejada: 3
Digite a chave a ser buscada: Marcos
Random Walk: Chave não encontrada ou TTL esgotado
Chave não encontrada

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
```

Figura 11 – TLL chegando a 0 na busca por 'random walk' - Topologia Ciclo 3

```
Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: 4
Digite a chave a ser buscada: Marcos
BP: TTL igual a zero, descartando mensagem
Chave não encontrada
```

Figura 12 – TLL chegando a 0 na busca por profundidade - Topologia Ciclo 3

Fora os casos onde o TTL chega a 0, podemos ver um exemplo na Figura 13 onde o TTL é decrementado usando o valor padrão (100), e encontra a chave desejada (pois não chegará a 0). Para evitar exibir o print de todas as buscas, o exemplo será exibido na busca por profundidade, mas todas funcionam de forma semelhante.

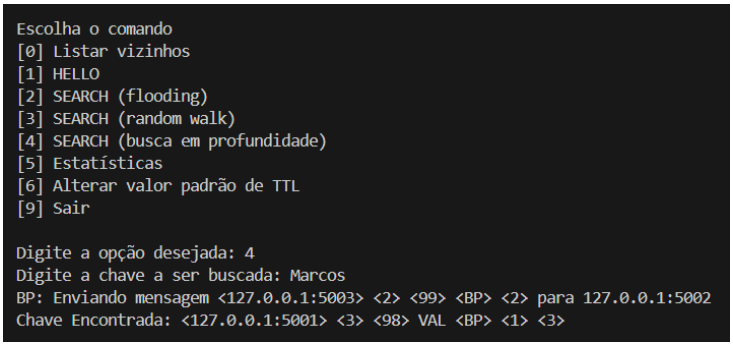

Figura 13 – TTL na busca por profundidade indo de 100 a 98

Cada busca tem seu TTL decrementado conforme as mensagens vão sendo encaminhadas, cumprindo o propósito da variável.

0.2.3 Busca por flooding

A busca por flooding é um dos tipos tipo de busca implementado no exercício programa, fizemos uma demonstração para algumas situações, sendo elas:

0.2.3.1 Nó encontrando uma chave que existe na rede

Usando uma topologia simples (ciclo 3) tentaremos encontrar a chave valor "SistemasDistribuidos 1", presente no peer 5002, a partir do peer 5003. A mensagem começa no peer 5003, como visto na Figura 14, passa para o peer 5001, na Figura 15, e por fim, encontra a chave no peer 5002, como pode-se ver na Figura 16.

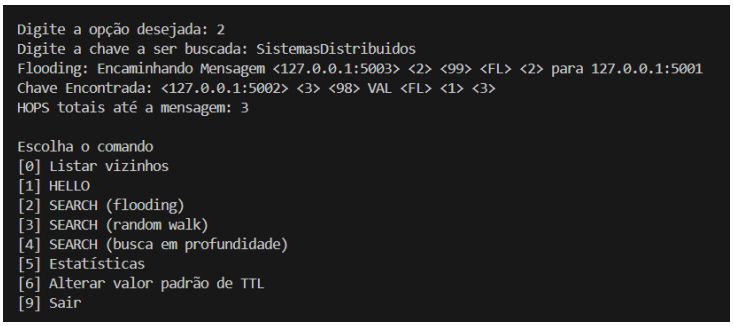

Figura 14 – Saída do peer 5003 - Flooding

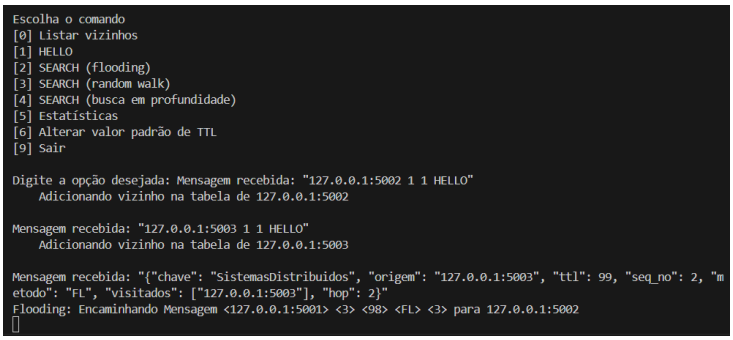

Figura 15 – Saída do peer 5001 - Flooding

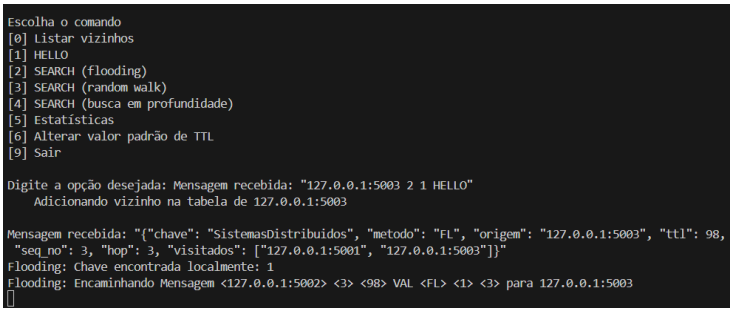

Figura 16 – Saída do peer 5002 - Flooding

0.2.3.2 Mensagens de busca por flooding não são re-enviadas para o nó remetente

Por meio do comando de estatísticas, que será explicado mais a frente, conseguimos fazer um exemplo de uma busca por flooding, que tem algumas informações printadas na saída do código. Dentre essas informações, temos o dado de de quantas vezes a mensagem é passada para o nó inicial nos 3 tipos de busca. Como vamos ver nos testes feitos usando as topologias "três triângulos"e "árvore binária"o resultado dessa estatística sempre será 0.

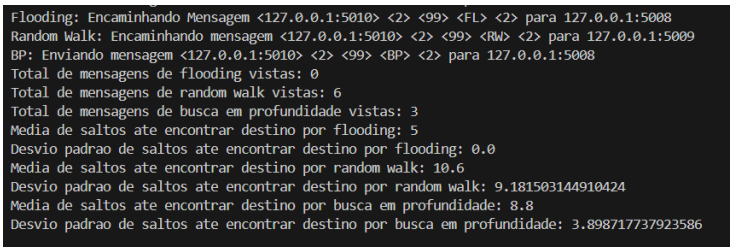

Figura 17 – Total de mensagens de flooding vistas resultando em 0 na topologia "três triângulos"buscando pela chave "Clara", iniciando no peer 5010

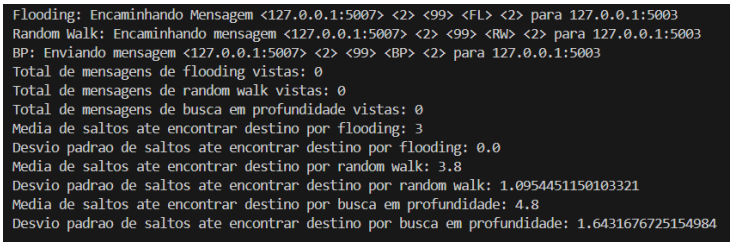

Figura 18 – Total de mensagens de flooding vistas resultando em 0 na topologia "árvore binária"buscando pela chave "Clara", iniciando no peer 5007

Além disso, no código foi implementado uma lista de peers visitados para evitar que nós processem a busca mais de uma vez, o que implica também que o nó remetente não terá sua busca re-enviada, já que ele é adicionado a essa lista antes de que a mensagem seja passada.

Figura 19 – Print do Código com a parte da logica descrita

0.2.3.3 Mensagens de busca por flooding repetidas são descartadas

O Código demonstrado na Figura 19 também é responsável por essa parte da busca, visto que ele adiciona o nó na lista de nós visitados. Para exemplificar isso, usaremos a topologia grid 3x3 e veremos um caso em que a mensagem não é passada para frente por conta disso.

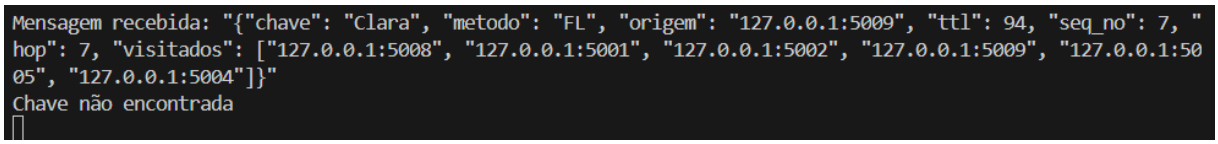

Figura 20 – Peer 5007 não passando a mensagem para frente pois todos os outros peers já haviam recebido a mensagem

Nesse caso, a busca pela chave "Clara"(presente no peer 5003) começou no peer 5009, mas em um certo ponto, ao chegar o peer 5007, todos os seus vizinhos (5004 e 5008) já haviam sido visitados.

0.2.4 Busca por random walk

Random walk é um tipo de busca implementada no exercício programa e que consegue encontrar uma chave na rede. Ela pode ser com ciclos ou sem ciclos, como demonstrada nas subseções a seguir.

0.2.4.1 Rede com ciclos

A rede com ciclo utilizada será a "grid 3x3". Queremos encontrar a chave "Clara", presente no peer 5003 a partir do peer 5009.

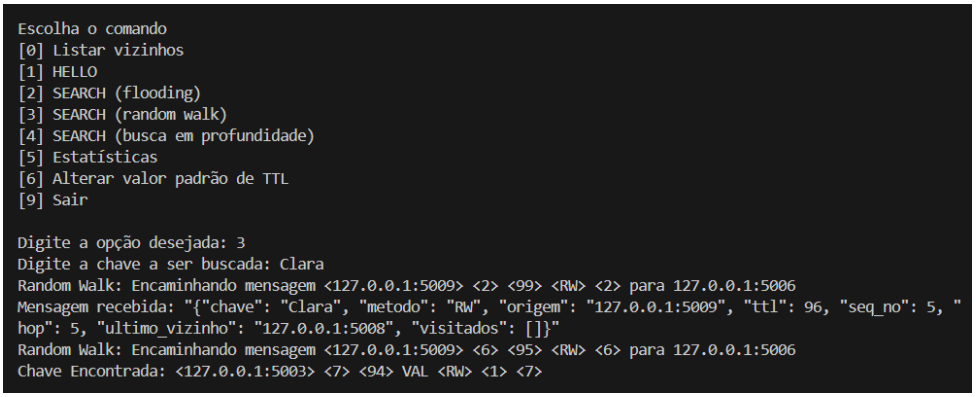

Figura 21 – Saída do peer 5009

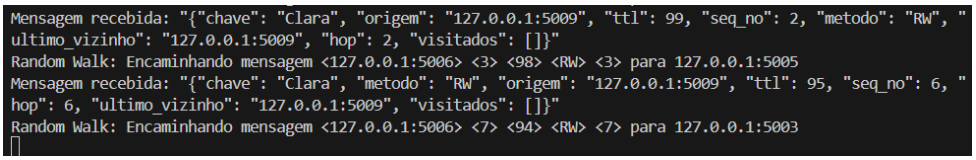

Figura 22 – Saída do peer 5006

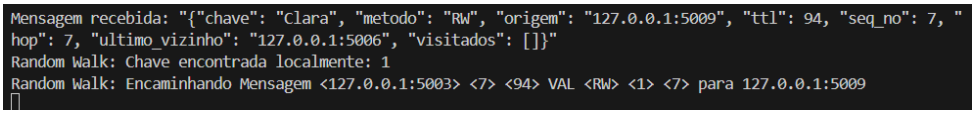

Figura 23 – Saída do peer 5003

Como pode-se ver nas Figuras, a mensagem da busca por random walk também é enviada para outros peers, mas sempre encontrando a chave.

0.2.4.2 Rede sem ciclos

A rede sem ciclo utilizada será a "árvore binária". Queremos encontrar a chave "SistemasDistribuidos", presente no peer 5002 a partir do peer 5007.

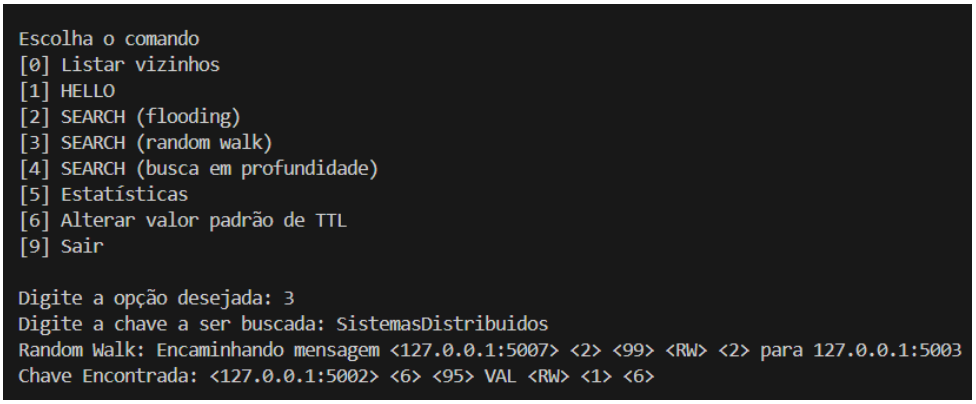

Figura 24 – Saída do peer 5007

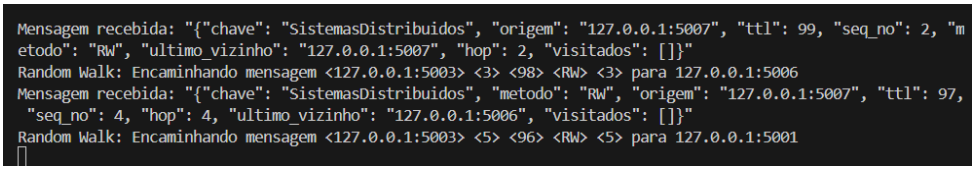

Figura 25 – Saída do peer 5003

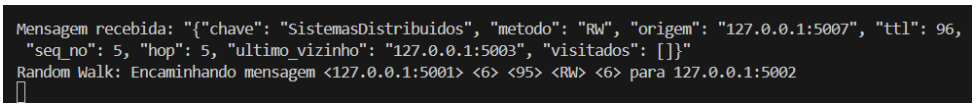

Figura 26 – Saída do peer 5001

Figura 27 – Saída do peer 5002

0.2.5 Busca em Profundidade:

Primeiramente será demonstrado sua capacidade de achar a chave numa topologia com ciclos (Topologia Três Triângulos)

Primeiramente, é feita a inicialização da rede (demonstraremos esse passo apenas uma vez):

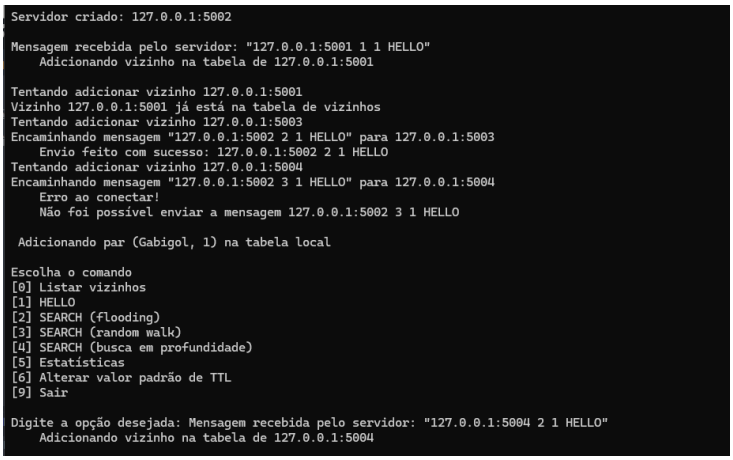

Figura 28 – Inicalização peer 5002

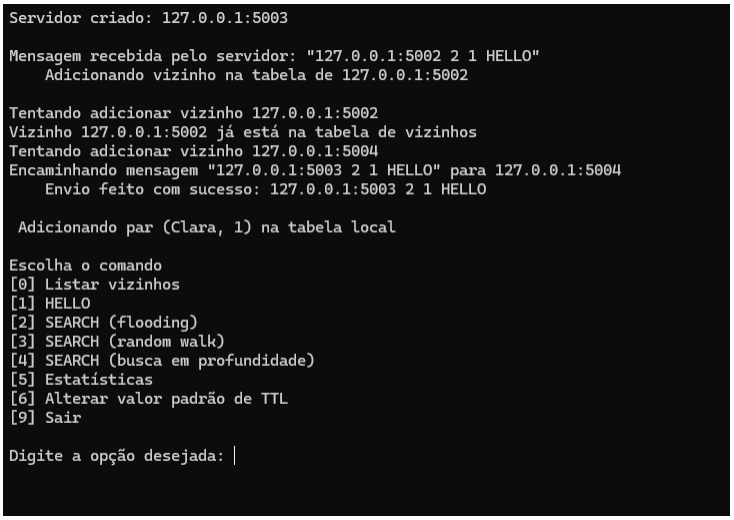

Figura 29 – Inicalização peer 5003

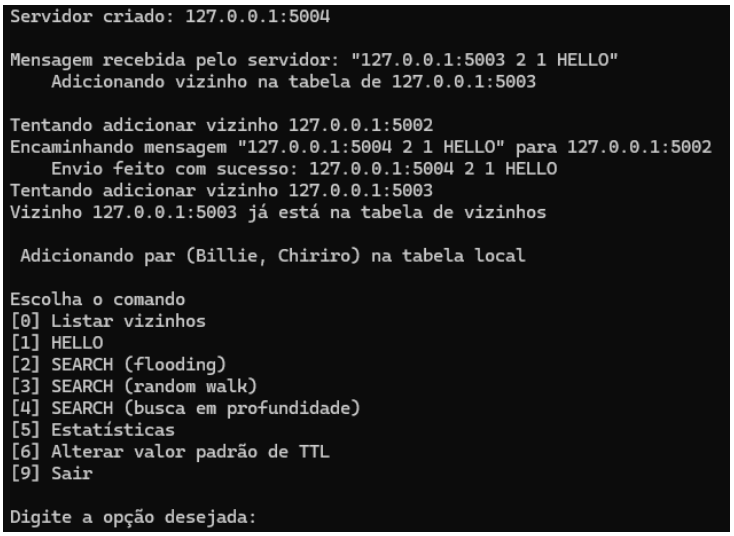

Figura 30 – Inicalização peer 5004

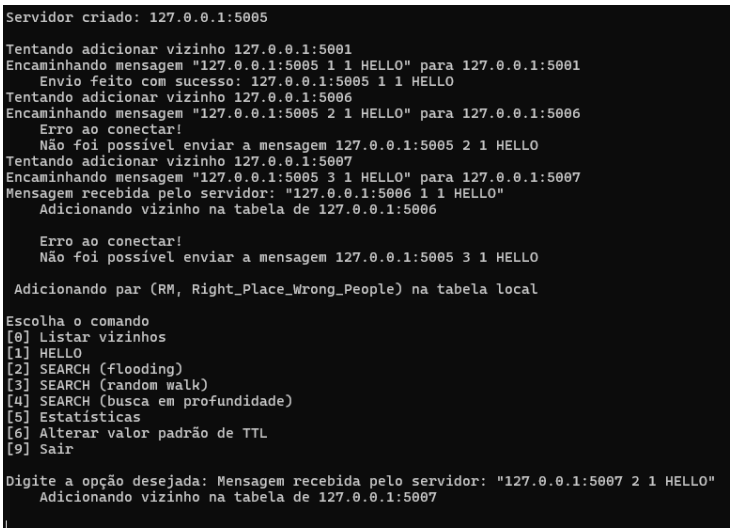

Figura 31 – Inicalização peer 5005

```
Servidor criado: 127.0.0.1:5006

Tentando adicionar vizinho 127.0.0.1:5005
Encaminhando mensagem "127.0.0.1:5006 1 1 HELLO" para 127.0.0.1:5005
    Envio feito com sucesso: 127.0.0.1:5006 1 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5007
Encaminhando mensagem "127.0.0.1:5006 2 1 HELLO" para 127.0.0.1:5007
    Envio feito com sucesso: 127.0.0.1:5006 2 1 HELLO

 Adicionando par (Aurora, Warrior) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada:
```

Figura 32 – Inicalização peer 5006

```
Servidor criado: 127.0.0.1:5007

Mensagem recebida pelo servidor: "127.0.0.1:5006 2 1 HELLO"
    Adicionando vizinho na tabela de 127.0.0.1:5006

Tentando adicionar vizinho 127.0.0.1:5005
Encaminhando mensagem "127.0.0.1:5007 2 1 HELLO" para 127.0.0.1:5005
    Envio feito com sucesso: 127.0.0.1:5007 2 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5006
Vizinho 127.0.0.1:5006 já está na tabela de vizinhos

 Adicionando par (Clarissa, 4AM) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada:
```

Figura 33 – Inicalização peer 5007

```
Servidor criado: 127.0.0.1:5008

Tentando adicionar vizinho 127.0.0.1:5001
Encaminhando mensagem "127.0.0.1:5008 1 1 HELLO" para 127.0.0.1:5001
    Envio feito com sucesso: 127.0.0.1:5008 1 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5009
Encaminhando mensagem "127.0.0.1:5008 2 1 HELLO" para 127.0.0.1:5009
    Erro ao conectar!
    Não foi possível enviar a mensagem 127.0.0.1:5008 2 1 HELLO
Tentando adicionar vizinho 127.0.0.1:5010
Encaminhando mensagem "127.0.0.1:5008 3 1 HELLO" para 127.0.0.1:5010
Mensagem recebida pelo servidor: "127.0.0.1:5009 1 1 HELLO"
    Adicionando vizinho na tabela de 127.0.0.1:5009

    Erro ao conectar!
    Não foi possível enviar a mensagem 127.0.0.1:5008 3 1 HELLO

 Adicionando par (EACH, USP) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: Mensagem recebida pelo servidor: "127.0.0.1:5010 1 1 HELLO"
    Adicionando vizinho na tabela de 127.0.0.1:5010
```

Figura 34 – Inicalização peer 5008

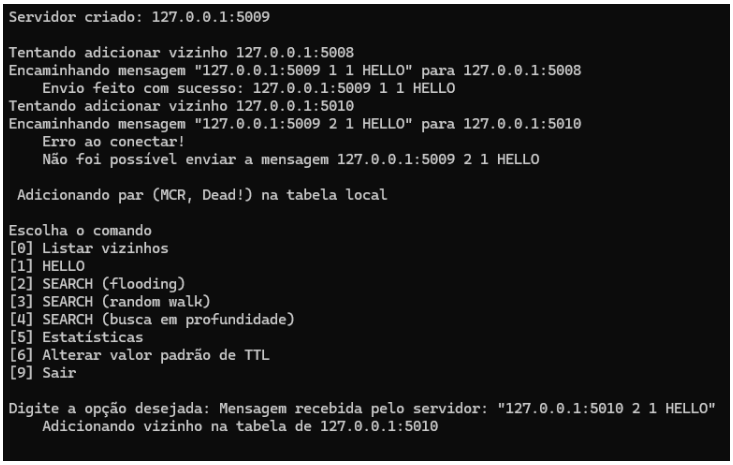

Figura 35 – Inicalização peer 5009

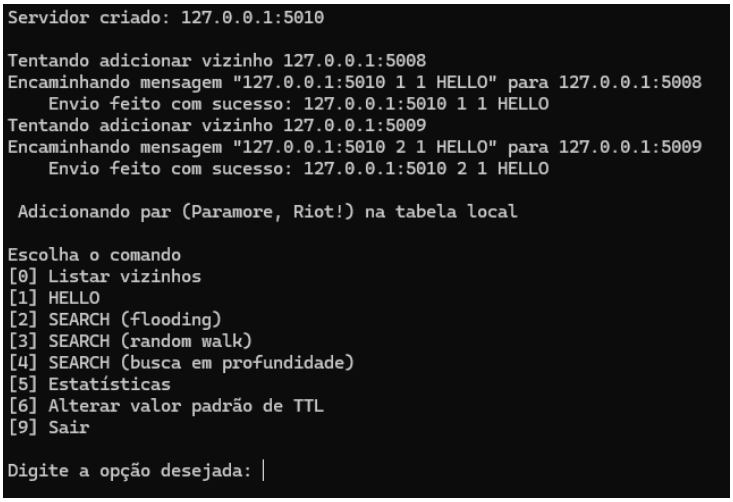

Figura 36 – Inicalização peer 5010

Logo em seguida, buscamos a chave "Aurora"(peer 5006) a partir do peer 5001:

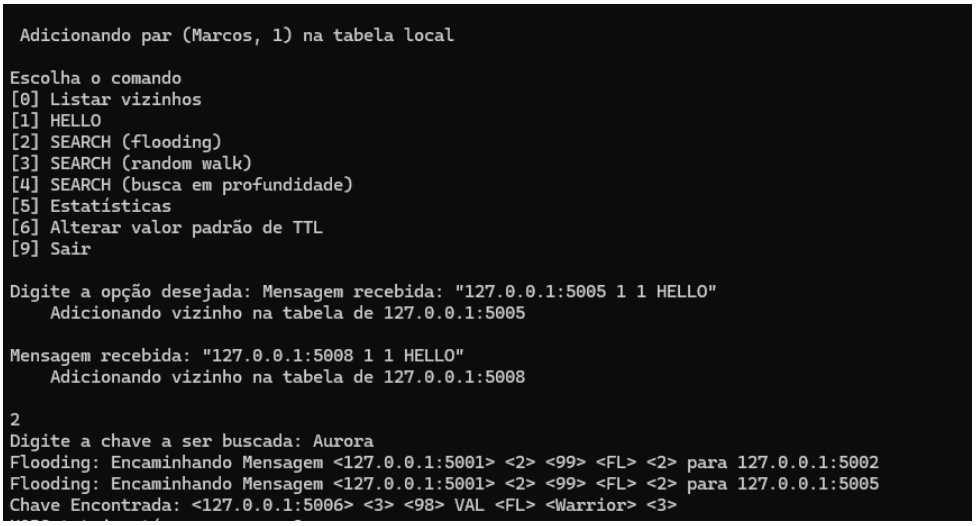

Figura 37 – Busca no terminal 5001

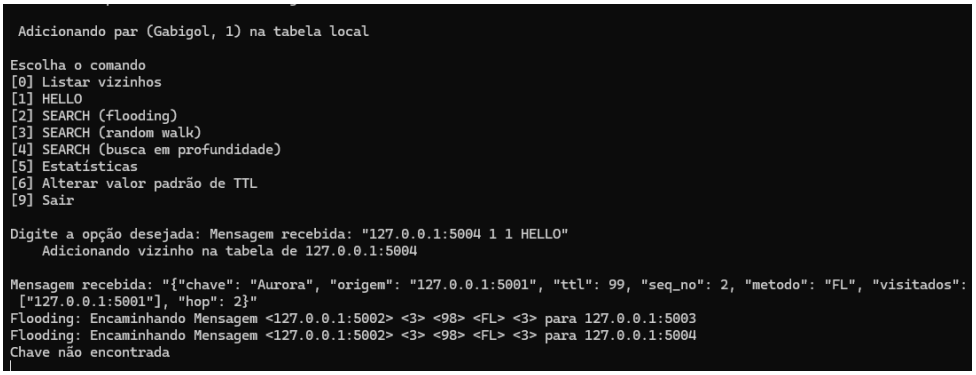

Figura 38 – Busca no terminal 5002

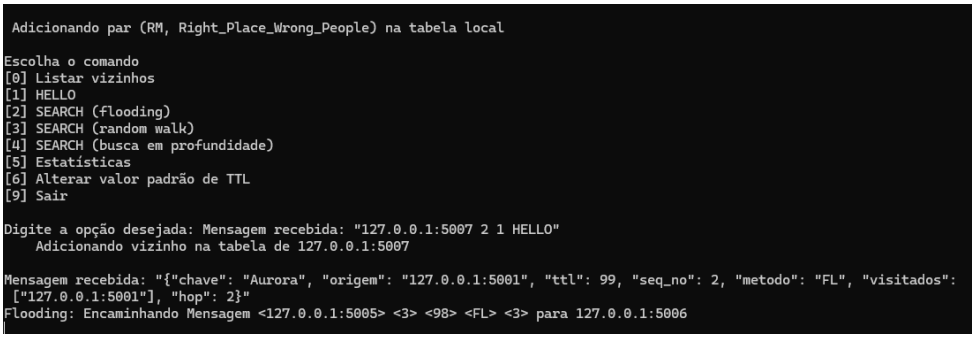

Figura 39 – Busca no terminal 5005

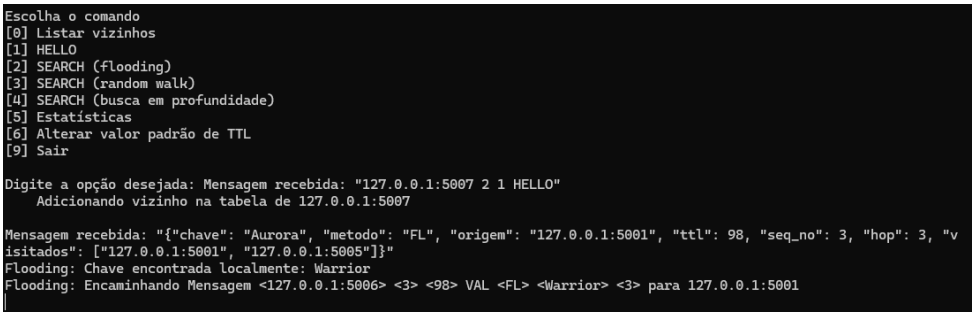

Figura 40 – Busca no terminal com a chave buscada 5006

Agora, será demonstrado sua capacidade de achar a chave numa rede sem ciclos (Topologia árvore binária):

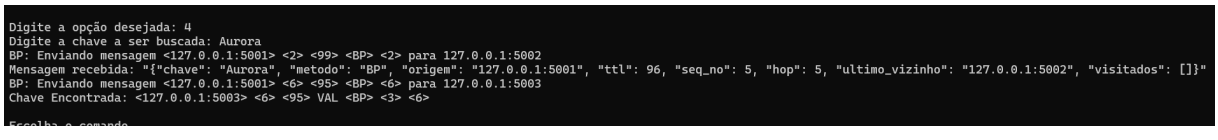

Figura 41 – Inicialização da busca no peer 5001

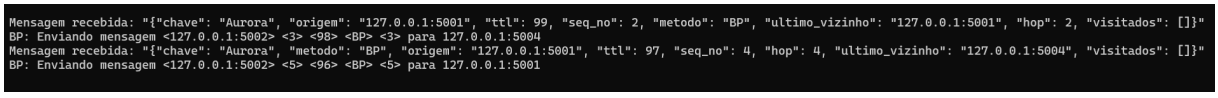

Figura 42 – Terminal peer 5002

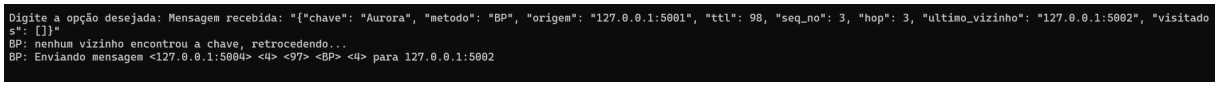

Figura 43 – Terminal peer 5004

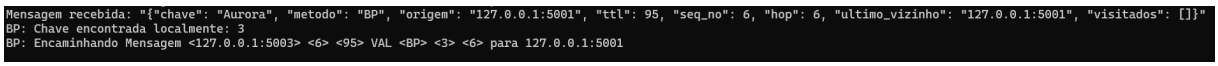

Figura 44 – Terminal peer 5003

0.2.6 Coleta de estatísticas

Para a função estatística, fazemos um for para realizar 5 execuções com cada busca para achar uma chave, e a partir dai geramos a média e desvio padrão para essas 5 execuções de cada busca, também é mostrada quantas mensagens de busca foram processadas pelo nó de origem na última execução

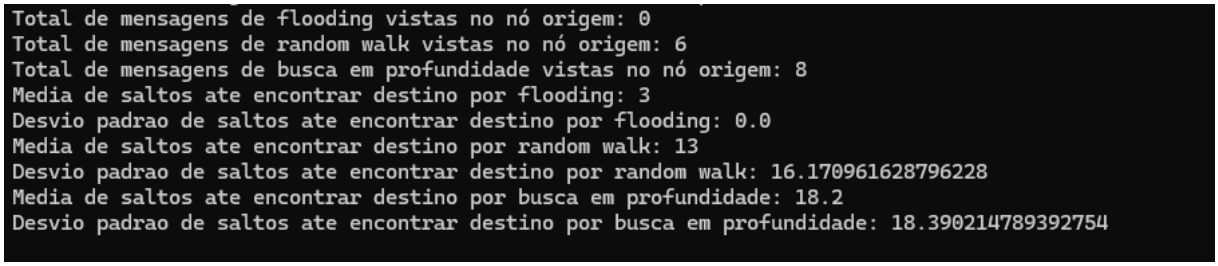

Figura 45 – Estatísticas para a busca na topologia de três triângulos

0.3 Análise de dados

A terceira parte do relatório envolve executar o programa em pelo menos dois computadores e em um topologia um pouco mais complexa (à escolha do grupo, com pelo menos 10 nós). Em seguida, escolher dois nós da rede para fazer as operações de busca algumas vezes (cada nó deve buscar sempre a mesma chave). Analise as estatísticas obtidas. Elas estão de acordo com o esperado?

0.4 Teste entre dois computadores na mesma rede

Para esse teste foi utilizado um desktop e um notebook conectados na mesma rede wifi, ambos com o firewall desligado. Foi utilizado a topologia de três triângulos, onde os peers 5006 e 5007 ficaram no notebook e os outros peers no desktop

Como é possível ver, os peers 5006 e 5007 estão em outro endereço ipv4, mas ainda sim a conexão foi possível

```
Servidor criado: 192.168.56.1:5005

Tentando adicionar vizinho 192.168.1.12:5006
Encaminhando mensagem "192.168.56.1:5005 1 1 HELLO" para 192.168.1.12:5006
    Envio feito com sucesso: 192.168.56.1:5005 1 1 HELLO
Tentando adicionar vizinho 192.168.1.12:5007
Encaminhando mensagem "192.168.56.1:5005 2 1 HELLO" para 192.168.1.12:5007
    Envio feito com sucesso: 192.168.56.1:5005 2 1 HELLO
Tentando adicionar vizinho 192.168.56.1:5001
Encaminhando mensagem "192.168.56.1:5005 3 1 HELLO" para 192.168.56.1:5001
    Envio feito com sucesso: 192.168.56.1:5005 3 1 HELLO

 Adicionando par (Freddie, 5) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: 0
Há 3 vizinhos na tabela:
[0] 192.168.1.12:5006
[1] 192.168.1.12:5007
[2] 192.168.56.1:5001
```

Figura 46 – Terminal 5005 fazendo conexão com 5006 e 5007 que estão em outra máquina

```
    Não foi possível enviar a mensagem 192.168.1.12:5006 1 1 HELLO
Tentando adicionar vizinho 192.168.1.12:5007
Encaminhando mensagem "192.168.1.12:5006 2 1 HELLO" para 192.168.1.12:5007
    Envio feito com sucesso: 192.168.1.12:5006 2 1 HELLO

 Adicionando par (Chorao, 6) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: Mensagem recebida: "192.168.56.1:5005 1 1 HELLO"
    Adicionando vizinho na tabela de 192.168.56.1:5005
```

Figura 47 – Terminal 5006

Prints de um hello bem sucedido entre os dois peers:

```
Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: 1
Há 3 vizinhos na tabela:
[0] 192.168.1.12:5006
[1] 192.168.1.12:5007
[2] 192.168.56.1:5001
Escolha o vizinho para enviar HELLO: 1
Encaminhando mensagem "192.168.56.1:5005 4 1 HELLO" para 192.168.1.12:5007
    Envio feito com sucesso: 192.168.56.1:5005 4 1 HELLO
```

Figura 48 – Terminal 5005

```
 Adicionando par (Gabigol, 7) na tabela local

Escolha o comando
[0] Listar vizinhos
[1] HELLO
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair

Digite a opção desejada: Mensagem recebida: "192.168.56.1:5005 2 1 HELLO"
    Adicionando vizinho na tabela de 192.168.56.1:5005

Mensagem recebida: "192.168.56.1:5005 4 1 HELLO"
    Vizinho 192.168.56.1:5005 já está na tabela de vizinhos
```

Figura 49 – Terminal 5007

Prints de uma busca por flooding na rede utilizando duas máquinas (a busca se inicia no 5001, que esta em uma máquina, e quer achar a chave "Chorao", que está no peer 5006, na outra máquina):

```
    Adicionando vizinho na tabela de 192.168.56.1:5008

2
Digite a chave a ser buscada: Chorao
Flooding: Encaminhando Mensagem <192.168.56.1:5001> <2> <99> <FL> <2> para 192.168.56.1:5002
Flooding: Encaminhando Mensagem <192.168.56.1:5001> <2> <99> <FL> <2> para 192.168.56.1:5005
Chave Encontrada: <192.168.1.12:5006> <3> <98> VAL <FL> <6> <3>
HOPS totais até a mensagem: 3
```

Figura 50 – Terminal 5001

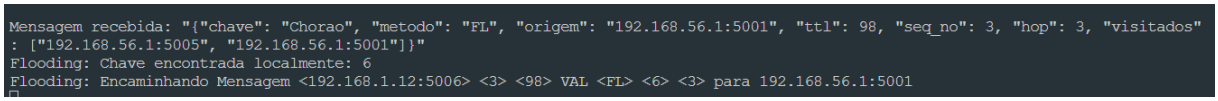

Figura 51 – Terminal 5006

Prints de uma busca por profundidade na rede utilizando duas máquinas (a busca se inicia no 5001, que esta em uma máquina, e quer achar a chave "Chorao", que está no peer 5006, na outra máquina):

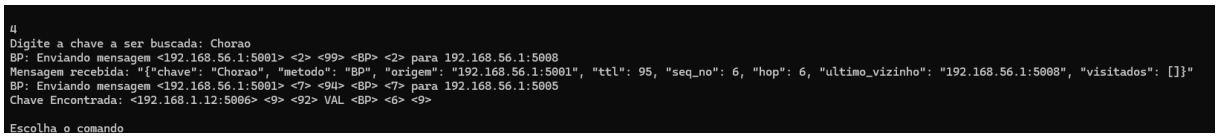

Figura 52 – Terminal 5001

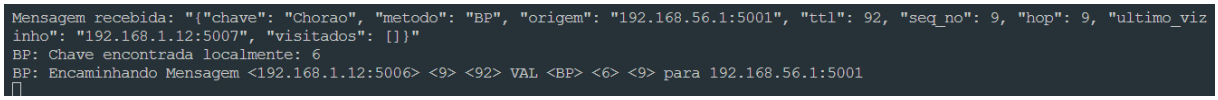

Figura 53 – Terminal 5006

Prints do comando estatística que realiza várias buscas pela rede, nota-se primeiramente que o comando é executado de forma um pouco mais lenta do que comparado a quando todos os peers estão em uma só máquina.

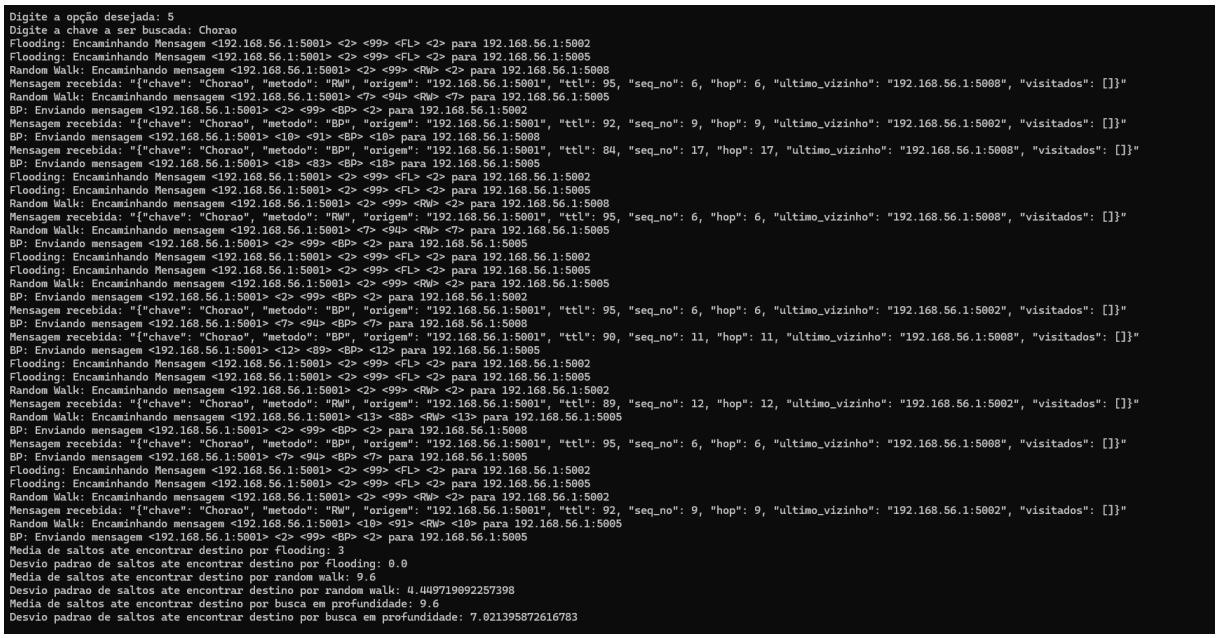

Figura 54 – Prints Estatística com rede em dois computadores

Mas ainda sim, os resultados para cada tipo de algoritmo continuam como o esperado em cada tipo de busca.