Universidade Federal de Minas Gerais – Belo Horizonte, MG – 2019/1
Instituto de Ciências Exatas – Departamento de Ciência da Computação

DCC023: Redes de Computadores
TP3 – Rede Peer-to-Peer com Chave-Valor

Aluna: Scarlet Gianasi Viana (2016006891)

1. Introdução

Este trabalho teve como objetivo implementar um sistema de armazenamento chave-valor entre pares, simulando uma rede *peer-to-per*, utilizando o protocolo TCP.

---

2. Arquitetura da Rede P2P

A rede implementada possui dois tipos de par: **clientes** e **nós** (*servents*). Os nós da rede formam uma topologia de rede sobreposta, com conexões fixas, e clientes podem se conectar a nós em portas específicas, para realizar requisições de chaves ou topologias.

2.1 Cliente

O cliente realiza uma conexão em uma porta a partir de um endereço de um *servent*. Com isto, o cliente pode enviar mensagens de requisição a partir do *servent*. Existem três comandos possíveis para o cliente:

- **`? <valor>`** → Consulta uma chave `<valor>`.
- **`T`** → Consulta a topologia da rede.
- **`Q`** → Finaliza o programa.

A inicialização do cliente se dá pelo comando:

```
./TP3client.py <PORTO_C> <IP:PORTO_S>
```

Em que `<PORTO_C>` é o porto do cliente e `<IP:PORTO_S>` é o endereço do *servent* ao qual o cliente irá se conectar. Os portos devem ser diferentes, e o servent já deve ter sido inicializado.

O programa **Cliente** recebe comandos do usuário (*stdin*), espera uma resposta por 4 segundos no máximo, imprime a resposta na tela (se recebida) e depois volta a receber comandos. Ao finalizar o programa, fecha sua conexão com o seu *servent*.

O cliente pode receber uma resposta de qualquer *servent* da rede. Respostas possíveis são o valor contido na chave de um banco armazenado em um nó, ou a topologia conhecida da rede de cada servent que recebeu uma mensagem de alagamento de topologia. Portanto, pode ser que o cliente receba mais de uma resposta por requisição. Neste caso, as respostas são todas impressas, se válidas. Respostas são válidas se seu número de sequência (**nseq**) for o mesmo da requisição.

Caso seja uma resposta de **consulta de chave**, a saída segue o seguinte modelo:

```
<valor> <IP_origem:PORTA_origem>
```

Em que `<valor>` é o valor encontrado a partir da chave dada na requisição, e `<IP_origem>` e `<PORTA_origem>` correspondem ao endereço do *servent* que enviou a resposta.

Caso seja uma resposta de consulta de topologia da rede, a saída segue a formatação:

```
<IP1:PORTO1> [<IP2:PORTO2> … <IPx:PORTOx>]
```

Em que `<IPn>:<PORTOn>` correspondem aos **n** endereços dos *servents* que passaram pelo *servent* que enviou a resposta, incluindo seu próprio endereço.

Caso nenhuma resposta para a requisição do cliente for recebida após 4 segundos sem receber mensagens, o cliente imprime na tela "**`Nenhuma resposta recebida.`**" e retorna a aceitar comandos do usuário para iniciar novas requisições. A cada resposta incorreta (número de sequência incorreto, por exemplo), o cliente imprime na tela "**`Mensagem incorreta recebida de <IP:PORTO>`**", com o endereço de origem da mensagem.

2.2 Servent (Nó)

O *servent* abre uma porta em um endereço e inicializa conexões com vizinhos (se dados na inicialização), inicializando a topologia da rede com outros *servents*. Todos os *servents* da topologia devem ser inicializados em ordem para que a rede seja criada com sucesso. Os servents armazenam um banco com chaves e valores correspondentes, e recebem requisições de clientes para consultar os valores contidos nas chaves.

Os servents também são responsáveis por enviar mensagens de alagamento de chave ou topologia para seus vizinhos, e mensagens de resposta (com valores ou topologias) para clientes. Sua inicialização se dá pelo seguinte comando:

```
./TP3node.py <PORTO> <ARQUIVO_BANCO> [<IP1:PORTO1> … <IPX:PORTOX>]
```

Em que `<PORTO>` é a porta que abrirá, `<ARQUIVO_BANCO>` é um arquivo de texto que possui o banco com chaves e valores, e cada `<IPn:PORTOn>` é o endereço de algum vizinho seu.

O primeiro servent da topologia não possui nenhum vizinho. O segundo possui o primeiro como vizinho. O terceiro possui um dos dois ou os dois como vizinhos. Ou seja, apenas poderão ser inicializados vizinhos quando o servent correspondente àquele endereço já foi inicializado.

O arquivo do banco segue a seguinte formatação:

```
# Comentário, descartado
chave1     valor 1       # Faz parte do valor
chave2     valor 2
…
```

Em que, para cada linha no arquivo, se for iniciada com “#”, é considerada um comentário, e descartada. Cada primeira palavra de uma linha que não seja um comentário é considerada uma chave, e tudo que vem depois desta palavra, incluindo espaços, tabulações e “#”s, até uma quebra de linha, é considerado o valor desta chave. Caso exista uma chave com mesmo nome no banco, seu valor é sobrescrito, ou seja, seu valor, ao final, será o último valor da chave com aquele nome.

No exemplo acima, o banco seria definido por estas chaves e valores:

```
banco = {
chave1: “     valor 1       # Faz parte do valor”,
chave2: “     valor 2”
}
```

Desta forma, o *servent* pode receber requisições de chave de clientes a ele conectados e buscar a chave em seu banco. Caso possua a chave, envia uma mensagem de resposta para o cliente com o valor da chave. Caso não possua, inicia o protocolo de alagamento, enviando mensagens para seus vizinhos, para que eles chequem seus bancos em busca da chave.

Servents também podem receber requisições de topologia de clientes. Ao rebecer uma requisição de topologia, o *servent* envia mensagens de alagamento de topologia para todos os seus vizinhos, com seu próprio endereço, para que, a cada servent que receber a mensagem, adicione seu próprio endereço também. A cada mensagem de alagamento (e requisição de alagamento), é enviada uma resposta para o cliente que iniciou a consulta com os valores naquele momento. Ou seja, um cliente vai receber várias mensagens de resposta para uma consulta de topologia, se existirem vários *servents* (alcançáveis) na rede.

2.3 Servent-servent - Alagamento

O protocolo de troca de mensagens de *servent* para *servent* se dá pela conexão de *servents* na topologia inicial da rede e pela troca de mensagens de identificação e alagamento. A cada conexão com um *servent*, uma mensagem de identificação é enviada (com valor 0 se vier de um servent e o valor do porto se vier de um cliente), e, a cada mensagem de requisição de chave ou topologia recebida de um cliente, o servent enviará mensagens de alagamento para todos os seus vizinhos (se a mensagem estiver “viva”). Seus vizinhos, ao receber a mensagem de alagamento, diminuem o tempo de vida da mensagem (começa com 3, em 0 a mensagem já “morre”), e, caso ainda esteja viva, retransmite a mensagem (adicionando seu próprio endereço) para todos seus vizinhos (exceto o que lhe enviou). Dessa forma, uma requisição de topologia pode “alagar” a rede, possibilitando o cliente enxergar a topologia da rede e então encontrar chaves em bancos de *servents* que não estão diretamente conectados a ele.

---

3. Mensagens

Para a identificação das mensagens, são definidos tipos com identificadores e campos específicos, a seguir:

Campo **TIPO**:

ID: 4	**KEYREQ**: 5	**TOPOREQ**: 6	**KEYFLOOD**: 7	**TOPOFLOOD**: 8	**RESP**: 9

Mensagens de requisição de chave, topologia, e de alagamento, assim como resposta, possuem um número de sequência, **nseq**, iniciado em 0 no cliente que inicia uma requisição, que identifica a consulta. Ou seja, quando um cliente faz uma consulta com um dado **nseq**, este nseq será distribuído nas requisições seguintes passadas pelos servents, para que as resposta da requisição possuam o identificador correto.

O formato dos campos das mensagens está a seguir, com os números abaixo dos campos representando o número de bytes que usam.

ID – Mensagem de identificação:

TIPO	PORTO
2	2

KEYREQ – Mensagem de requisição de chave:

TIPO	NSEQ	TAMANHO	CHAVE
2	4	2	1 … 400

TOPOREQ – Mensagem de requisição de topologia:

TIPO	NSEQ
2	4

KEYFLOOD ou TOPOFLOOD – Mensagem de alagamento de chave ou topologia:

TIPO	TTL	NSEQ	IP_ORIG	PORTO_ORIG	TAMANHO	INFO
2	2	4	4	2	2	1 … 400

RESP – Mensagem de resposta:

TIPO	NSEQ	TAMANHO	VALOR
2	4	2	1 … 400

---

4. Implementação

A implementação deste trabalho foi feita utilizando a linguagem **Python** (versão 3.6.7), utilizando as bibliotecas (para o uso do protocolo TCP) **socket** e **select**, em especial. A descrição da implementação de **TP3client.py** e **TP3node.py** estão a seguir.

4.1 Implementação do Client

A implementação do cliente se deu primeiramente pela criação do soquete para conexão com o *servent*, no endereço dado como argumento na linha de comando. A partir da conexão, o cliente envia uma mensagem ID para o *servent*, com sua porta, para que possa ser identificado como cliente.

Então, inicia-se o loop de consultas, esperando comandos do usuário no terminal.

- Se o comando for para uma consulta de chave, uma mensagem do tipo KEYREQ é criada com o número de sequência (inicialmente 0) atual, e a mensagem é enviada pelo soquete.
  - **nseq** é atualizado, somando mais uma unidade.
  - Após o envio da mensagem, um novo loop se inicia, esperando o recebimento da resposta. Isso se deu pelo estabelecimento de um **timeout** de 4 segundos, em que o cliente espera aceitar conexões e receber mensagens de *servents*.
    - Se nenhuma mensagem for recebida em 4 segundos, é impresso "`Nenhuma resposta recebida`.", o loop é fechado, e o programa volta ao loop de consultas.

  - Se uma mensagem foi recebida, ela é avaliada:
    - Se é uma mensagem de resposta, com número de sequência correto, seu valor é impresso na tela.
    - Se for uma mensagem com sequência correta ou de outro tipo que não resposta, uma mensagem "`Mensagem incorreta recebida de <IP:PORTO>`", com o endereço de origem da mensagem, é impressa. Como o programa só sai deste loop após 4 segundos, outras mensagens, além das esperadas, podem ser recebidas e terem seus resultados impressos na saída.

Este é o funcionamento geral do cliente, e os métodos nele implementados são relacionados ao envio e recebimento de mensagens num soquete TCP. O recebimento, em especial, depende do tipo da mensagem, que define seu tamanho. Como no TCP as mensagens são enviadas como um *byte stream*, é necessário tomar muito cuidado ao receber cada campo da mensagem. Devido a isto, e como não foi especificado um campo de início ou término de mensagem, se assume neste trabalho que todas as mensagens enviadas não possuem erros relacionados aos campos e tamanho dos mesmos.

Outros métodos importantes implementados no cliente foram os de criação de mensagens (também há criação de mensagens no servent, mas de outros tipos – exceto ID, que é comum aos dois) de requisição de chave e topologia. Para a criação destas mensagens, foram utilizados structs para empacotar os bytes da forma correta, de maneira que seu recebimento possa ser concluido sem erros.

4.2 Implementação do Servent

Inicializando o *servent*, o banco de chave-valor é lido do arquivo e armazenado em um dicionário. O *servent* abre uma porta no endereço dado pela linha de comando e, caso tenha vizinhos dados como argumento (há no máximo 10 vizinhos), conecta com eles, criando um soquete para cada um, e os adicionando na lista de inputs para que possam enviar e receber mensagens.

Para esta troca de mensagens em múltiplos soquetes foi utilizada a função **select** da biblioteca de mesmo nome, que indica os soquetes que estão prontos para receber mensagens. Dessa forma, toda vez que um vizinho se conectar com o *servent*, a conexão pode ser aceita e o *servent* pode receber e enviar mensagens para este vizinho, criando uma topologia de rede.

A cada mensagem que o *servent* receber, dependendo de seu tipo, sua leitura é feita.

- Se a mensagem recebida for um ID, o servent pode identificar se a conexão é de um cliente ou outro *servent*. Dessa maneira, pode salvar os endereços e distinguir quem é servent e quem é cliente, auxiliando no envio de mensagens de cada tipo.

- Se a mensagem recebida for uma requisição de chave, *o servent* busca a chave em seu banco.
  - Se ele tiver a chave, ele envia uma mensagem RESP para o cliente que enviou a requisição, realizando uma conexão rápida com o mesmo em seu endereço.
  - Se ele não tiver a chave, ele inicia o protocolo de alagamento e envia uma mensagem KEYFLOOD para todos os seus vizinhos, com o endereço do cliente que requisitou a chave e com TTL igual a 3, inicialmente.

- Se a mensagem recebida for a mensagem de alagamento de chave, checa se TTL é maior que zero.
  - Se não for, descarta a mensagem.
  - Se for, checa se possui a chave requisitada no banco.
    - Se possuir, se conecta ao cliente cujo endereço se encontra na mensagem, e envia a ele uma mensagem de resposta com o valor da chave no banco.
    - Se não possui a chave, retransmite a mensagem com TTL subtraído uma unidade para todos seus vizinhos, exceto o vizinho que lhe enviou a mensagem.

- Se receber uma mensagem de alagamento de topologia, checa se TTL é maior que zero.
  - Se não, descarta a mensagem.
  - Se sim, adiciona seu endereço no campo de valor da mensagem e retransmite ela para todos seus vizinhos, exceto o vizinho que lhe enviou a mensagem.
    - Também se conecta com o cliente original que requisitou a topologia e envia uma mensagem de resposta com o mesmo campo de valor da mensagem que retransmitiu.

- Se a mensagem recebida for uma mensagem de requisição de topologia, envia uma mensagem de alagamento de topologia com seu endereço no campo de valor para todos os seus vizinhos, com TTL inicial 3, mesmo número de sequência que a requisição, e com o endereço do cliente que a requisitou. Também envia uma mensagem de resposta para o cliente, mas apenas com seu endereço.

---

5. Testes

Alguns testes foram realizados para testar o funcionamento dos programas implementados. A seguir estão algumas topologias e resultados respectivos de alguns destes testes.

Teste 1: Topologia simples, 1 servent na porta 9094 e um cliente na porta 9095. O cliente requisitou uma chave presente no banco do servent, a topologia da rede e uma chave que não existe no banco. Depois, finalizou o programa. A especificação define apenas a saída do cliente, então algumas mensagens são impressas no servent para que seja possível avaliar seu estado. A seguir estão as saídas do programa:

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3client.py 9095 127.0.0.1:9094
? tcpmux
          1/tcp     # TCP Port Service Multiplexer 127.0.0.1:38628
T
127.0.0.1:9094 127.0.0.1:38630
? tcp
Nenhuma resposta recebida
Q
```

Teste 1: Cliente

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3node.py 9094 exemplo.txt
-- SERVENT - Porta: 9094 --
Nova conexão em ('127.0.0.1', 34392)
Recebeu IDmsg de cliente no porto 9095
Servent possui a chave. Enviando resposta para cliente no porto 9095
Conectando com o cliente na porta 9095
Requisição de topologia recebida. Iniciando alagamento.
Conectando com o cliente na porta 9095
Servent não possui a chave. Iniciando alagamento.
Fechando conexão em ('127.0.0.1', 34392)
```

Teste 1: Servent

É possível perceber que o cliente requisita uma chave, o servent encontra a chave no banco e envia para o cliente. Quando o cliente requisita a topologia, o servent envia seu endereço e inicia o alagamento, mas como é o único servent da rede, não envia mensagens a vizinhos. Quando o cliente requisita uma chave que o servent não possui, ele tenta iniciar o alagamento, mas, como é o único servent da rede, também não envia mais mensagens. Quando o cliente encerra sua conexão, o servent identifica esta ação e retira o cliente da sua lista de entradas.

Teste 2: O segundo teste foi um pouco mais complexo. Possui um cliente em um *servent*, dois *servents* no total. Cliente na porta 9095, seu *servent* na porta 9094, e o vizinho deste *servent* na porta 9092.

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3client.py 9095 127.0.0.1:9094
? tcpmux
          1/tcp     # TCP Port Service Multiplexer 127.0.0.1:38746
T
127.0.0.1:9094 127.0.0.1:38752
127.0.0.1:9094 127.0.0.1:9092 127.0.0.1:38754
? tcp
Nenhuma resposta recebida
? nbp
             2/ddp     # Name Binding Protocol 127.0.0.1:38760
a
Comando desconhecido
Q
```

Teste 2: Cliente (9095)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3node.py 9094 exemplo.txt
-- SERVENT - Porta: 9094 --
Nova conexão em ('127.0.0.1', 34508)
Recebeu ID de cliente no porto 9095
Nova conexão em ('127.0.0.1', 34510)
Recebeu ID de servent em ('127.0.0.1', 34510)
Servent possui a chave. Enviando resposta para cliente no porto 9095
Conectando com o cliente no porto 9095
Requisição de topologia recebida. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 34510)
Conectando com o cliente no porto 9095
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 34510)
Servent possui a chave. Enviando resposta para cliente no porto 9095
Conectando com o cliente no porto 9095
Encerrando conexão em ('127.0.0.1', 34508)
```

Teste 2: Servent (9094)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3node.py 9092 exemplo.txt 127.0.0.1:9094
-- SERVENT - Porta: 9092 --
Conectado ao vizinho 127.0.0.1:9094
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9095
Conectando com o cliente no porto 9095
Servent não possui a chave. Iniciando alagamento.
```

Teste 2: Servent (9092)

Nestes testes, é possível verificar as mensagens de alagamento de chave e topologia. Quando o cliente requisita uma chave que seu *servent* não possui, o servent envia uma mensagem para o seu vizinho, que não a possui. Então, após 4 segundos, o cliente imprime "`Nenhuma resposta recebida`". Quando o cliente requisita a topologia, recebe resposta dos dois *servents*.

Teste 3: O terceiro teste possui 3 clientes, 2 em um *servent*, 1 em outro *servent*. Os dois *servents* são vizinhos. As portas usadas são: cliente 9093 e cliente 9092 do *servent* 127.0.0.1:9090, cliente 9094 do *servent* 127.0.0.1:9091. Novamente, o *servents* são vizinhos.

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3node.py 9090 exemplo.txt
-- SERVENT - Porta: 9090 --
Nova conexão em ('127.0.0.1', 37282)
Recebeu ID de servent em ('127.0.0.1', 37282)
Nova conexão em ('127.0.0.1', 37284)
Recebeu ID de cliente no porto 9093
Requisição de topologia recebida. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 37282)
Conectando com o cliente no porto 9093
Nova conexão em ('127.0.0.1', 37290)
Recebeu ID de cliente no porto 9092
Requisição de topologia recebida. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 37282)
Conectando com o cliente no porto 9092
Servent possui a chave. Enviando resposta para cliente no porto 9092
Conectando com o cliente no porto 9092
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 37282)
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 37282)
Servent possui a chave. Enviando resposta para cliente no porto 9094
Conectando com o cliente no porto 9094
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9094
Conectando com o cliente no porto 9094
Encerrando conexão em ('127.0.0.1', 37284)
Encerrando conexão em ('127.0.0.1', 37290)
^C
Finalizando o servidor.
```

Teste 3: Servent (9090)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3node.py 9091 exemplo2.txt 127.0.0.1:9090
-- SERVENT - Porta: 9091 --
Conectado ao vizinho 127.0.0.1:9090
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9093
Conectando com o cliente no porto 9093
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9092
Conectando com o cliente no porto 9092
Servent possui a chave. Enviando resposta para cliente no porto 9093
Conectando com o cliente no porto 9093
Servent não possui a chave. Iniciando alagamento.
Nova conexão em ('127.0.0.1', 48248)
Recebeu ID de cliente no porto 9094
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 9090)
Requisição de topologia recebida. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 9090)
Conectando com o cliente no porto 9094
Encerrando conexão em ('127.0.0.1', 48248)
Encerrando conexão em ('127.0.0.1', 9090)
^C
Finalizando o servidor.
```

Teste 3: Servent (9091)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3client.py 9093 127.0.0.1:9090
T
127.0.0.1:9090 127.0.0.1:46372
127.0.0.1:9090 127.0.0.1:9091 127.0.0.1:46374
? nbp
             2/ddp     # Name Binding Protocol 127.0.0.1:46384
? t
Nenhuma resposta recebida
Q
```

Teste 3: Cliente (9093)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3client.py 9092 127.0.0.1:9090
T
127.0.0.1:9090 127.0.0.1:57340
127.0.0.1:9090 127.0.0.1:9091 127.0.0.1:57342
? tcpmux
           1/tcp     # TCP Port Service Multiplexer 127.0.0.1:57344
Q
```

Teste 3: Cliente (9092)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3 TP3client.py 9094 127.0.0.1:9091
? tcpmux
           1/tcp     # TCP Port Service Multiplexer 127.0.0.1:43878
T
127.0.0.1:9091 127.0.0.1:43880
127.0.0.1:9091 127.0.0.1:9090 127.0.0.1:43882
Q
```

Teste 3: Cliente (9094)

Os clientes requisitaram a topologia com sucesso em todos os casos, e conseguiram acessar chaves em *servents* diferentes (chave tcpmux estava em um servent e chave nbp em outro).

Teste 4: Este último teste buscou testar a funcionalidade do TTL das mensagens de alagamento. Para isso, foi criada uma topologia linear com 4 *servents*, e um cliente no primeiro deles.

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3
 TP3node.py 9090 exemplo.txt
-- SERVENT - Porta: 9090 --
Nova conexão em ('127.0.0.1', 56862)
Recebeu ID de servent em ('127.0.0.1', 56862)
Nova conexão em ('127.0.0.1', 56870)
Recebeu ID de cliente no porto 9092
Requisição de topologia recebida. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 56862)
Conectando com o cliente no porto 9092
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 56862)
```

Teste 4: Servent (9090)

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3
 TP3node.py 9091 exemplo2.txt 127.0.0.1:9090
-- SERVENT - Porta: 9091 --
Conectado ao vizinho 127.0.0.1:9090
Nova conexão em ('127.0.0.1', 48476)
Recebeu ID de servent em ('127.0.0.1', 48476)
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9092
Enviando mensagem de alagamento para ('127.0.0.1', 48476)
Conectando com o cliente no porto 9092
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 48476)
```

Teste 4: Servent (9091), vizinho de 9090

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3
 TP3node.py 9093 exemplo2.txt 127.0.0.1:9091
-- SERVENT - Porta: 9093 --
Conectado ao vizinho 127.0.0.1:9091
Nova conexão em ('127.0.0.1', 57730)
Recebeu ID de servent em ('127.0.0.1', 57730)
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9092
Enviando mensagem de alagamento para ('127.0.0.1', 57730)
Conectando com o cliente no porto 9092
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 57730)
```

Teste 4: Servent (9093), vizinho de 9091

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3
 TP3node.py 9094 exemplo2.txt 127.0.0.1:9093
-- SERVENT - Porta: 9094 --
Conectado ao vizinho 127.0.0.1:9093
Nova conexão em ('127.0.0.1', 56044)
Recebeu ID de servent em ('127.0.0.1', 56044)
Mensagem de alagamento recebida. Enviando resposta para cliente no porto 9092
Enviando mensagem de alagamento para ('127.0.0.1', 56044)
Conectando com o cliente no porto 9092
Servent não possui a chave. Iniciando alagamento.
Enviando mensagem de alagamento para ('127.0.0.1', 56044)
```

Teste 4: Servent (9094), vizinho de 9093

```
scarletgv@rosequartz:~/Downloads/Documents/UFMG/Redes/TP3/codigosfinais$ python3
 TP3node.py 9095 exemplo2.txt 127.0.0.1:9094
-- SERVENT - Porta: 9095 --
Conectado ao vizinho 127.0.0.1:9094
```

Teste 4: Servent (9095), vizinho de 9094

O cliente somente recebeu mensagens com a topologia da rede até o terceiro *servent* depois do seu próprio *servent*, como esperado. As mensagens de alagamento de chave e topologia não chegaram no último *servent* da rede, até lá o TTL da requisição já foi zerado.

---

6. Conclusão

Com este trabalho, foi possível implementar uma rede relativamente complexa, capaz de receber múltiplas conexões com clientes e topologias diferentes. Com isto, foi possível colocar em prática um pouco da camada de transporte e aplicação, além de métodos mais robustos de lidar com soquetes TCP. do A partir dos testes, foi possível testar o sucesso da implementação, que aceita múltiplos clientes e servents simultaneamente, enviando mensagens de alagamento de chave e topologia com um tempo de vida pela rede.