

# Trabalho Prático 3

Carolina Santos

Thiago Silva

## 1 Introdução

Neste trabalho desenvolvemos um sistema peer-to-peer. Desenvolvemos dois tipos de programas, um *servent* e um cliente. O Servent contém a lista de pares chave-valor e controla a troca de mensagens, e será explicado com mais detalhes na seção 3. Essa base de entradas chave-valor fica armazenada em um arquivo, que é informado na execução. O programa cliente fica responsável por exibir os valores para as chaves solicitadas pelo usuário, que devem ser consultados.

Para as consultas, utilizamos o protocolo de comunicação de alagamento confiável. É um protocolo simples definido na especificação.

## 2 Arquitetura

Para o desenvolvimento desse projeto, implementamos dois tipos de código e uma topologia específica para testá-los. Os códigos são de cliente e servent, que serão explicados em detalhe nas seções que seguem. A topologia foi feita baseada no exemplo dado, tendo cinco servents conectados "em série" e um cliente cujo ponto de contato é o primeiro servent. Uma ilustração da topologia pode ser vista na figura 2. A seguir, explicamos com mais detalhes a implementação do servent e do cliente.

## 3 Servent

Os servents trocam mensagens entre si, e quando são um ponto de contato, recebem mensagem do cliente e após montarem a query, propagam para seus vizinhos. Ao serem iniciados, os servents recebem o número de seu porto, o nome do arquivo local de dados de chave-valor e os endereços (ip e porto). O servent então espera pelo recebimento de mensagens. Se for do cliente, ele monta a query e envia através do alagamento confiável para os

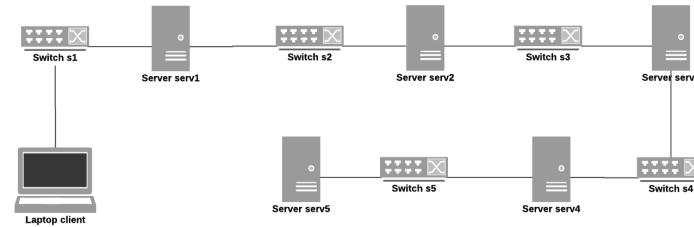


Figura 1: Topologia

outros servents. Caso a mensagem recebida já seja uma query, ele verifica em seu arquivo local se ele possui uma entrada para essa chave. Caso possua, ele envia a resposta diretamente para o cliente. Ele também propaga a mensagem para seus vizinhos, realizando o alagamento confiável.

O servent também mantém uma lista das mensagens recebidas, que é verificada sempre antes de se enviar a mensagem para os outros. Isso impede que o envio de mensagens fique descontrolado e em loop.

## 4 Cliente

O cliente é responsável por receber chaves do usuário, consultá-las e imprimir a resposta para o usuário. Cada cliente tem um ponto de contato, que é um nó na rede sobreposta, e quando a consulta deve ser feita, uma requisição com a consulta é enviada a esse ponto de contato.

O cliente, no momento da execução, recebe como parâmetro o endereço do seu ponto de contato, e quando se conecta, define o timeout para o socket. Quando o pacote com a consulta é enviado corretamente, as respostas são impressas até que o socket tenha seu timeout. Se ocorrerem dois timeouts o cliente não tenta enviar a mensagem novamente. A mensagem é composta do tipo da mensagem e o texto da chave.

## 5 Discussão

Nesse projeto, o cliente manda uma solicitação ao seu ponto de contato, que monta a query de busca, preenchendo o endereço de quem solicitou a consulta, incrementando o número de sequência e enviando a mensagem para

todos seus vizinhos. Ao mesmo tempo ele busca a chave no seu arquivo local e envia o que tiver ao cliente. Ao enviar uma resposta ao cliente, ele subtrai o TTL, e somente se o TTL for maior que zero ele propaga a mensagem aos seus vizinhos.

Quando o servernt possui a chave em seu arquivo local, ele gera uma resposta direta para o cliente, buscando seu endereço nos campos da query (onde o ponto de contato do cliente solicitante colocou seu endereço).

Uma grande parte da implementação foi garantir que não ocorram loops infinitos na propagação de mensagens, ou que os servernts mandem mensagens descontroladamente. Para resolver parte desses problemas, cada servernt mantém uma lista das mensagens que já foram recebidas, para que não fiquem repetindo e reenviando mensagens que já passaram por ele.

## 6 Execução

Para executar o sistema de mensagens, executamos o mininet com o comando "sudo mn -x -link tc -custom src/topology.py -topo mytopo", onde o arquivo topology.py, disponível em src/topology.py, contém a topologia a ser utilizada pelo mininet (a mesma exemplificada na especificação do trabalho, com valores de delay e banda como default).

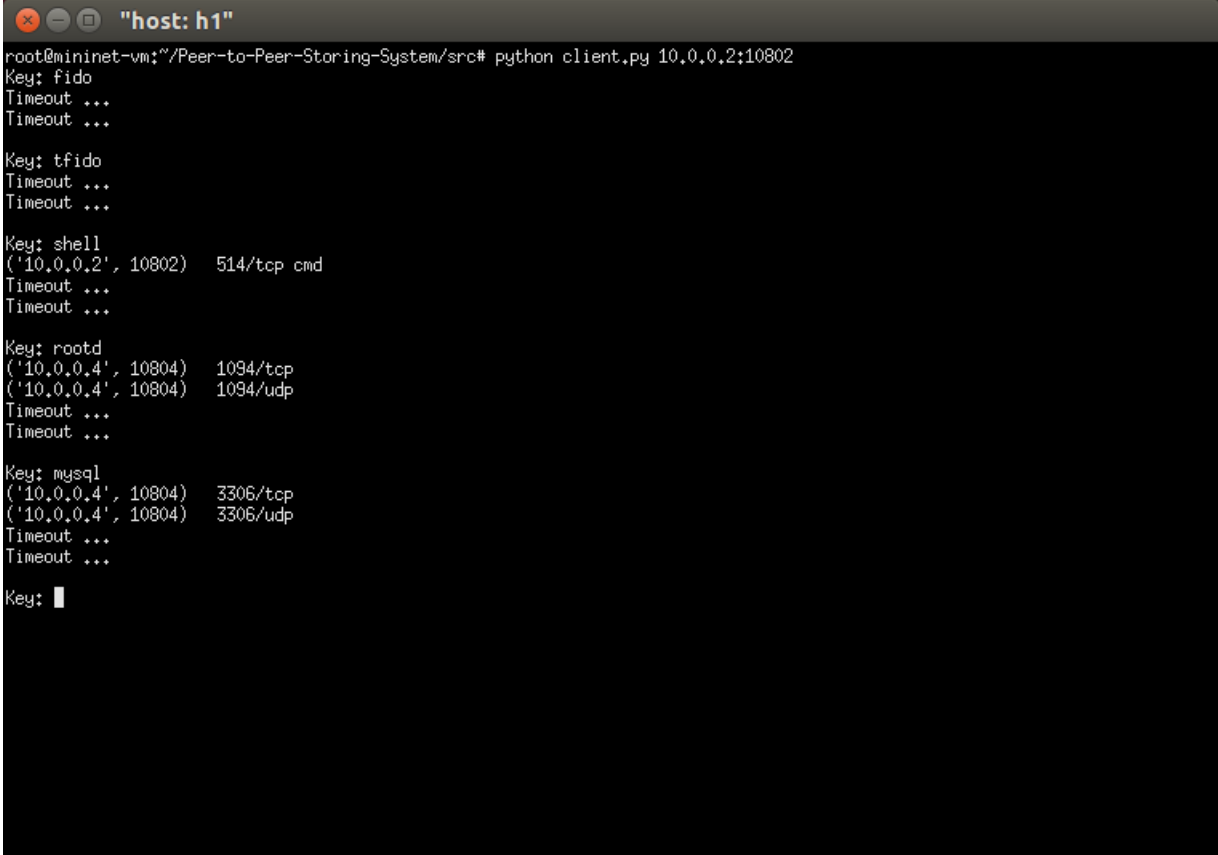
Os hosts criados pelo mininet tem sua função definidos pela topologia. O host 1 é o cliente por default e para iniciá-lo, executamos "python src/client.py ip-servernt:porto-servernt", onde ip-servernt é o ip do servernt que é seu ponto de contato e porto-servernt é o porto do mesmo.

Os hosts de 2 a 6 representam os servernts, que estão conectados "em série". Para inciar cada um executamos "python src/servernt.py meu-porto nome-do-arquivo.txt ip1:porto1 ip2:porto2 ip3:porto3 ...", onde meu-porto é o número do porto que está sendo iniciado, o arquivo txt é o arquivo que possui os dados de chave-valor, e os pares de ip e porto são os pares vizinhos ao servernt sendo iniciado. Dessa forma, o servernt serv2 por exemplo, teria apenas o ip e porto dos servernts serv1 e serv2.

A topologia pode ser alterada, e podemos executar quantos servernts e clientes quanto desejado.

## 7 Testes

### 7.1 Cliente



```
root@mininet-vml:/Peer-to-Peer-Storing-System/src# python client.py 10.0.0.2:10802
Key: fido
Timeout ...
Timeout ...

Key: tfido
Timeout ...
Timeout ...

Key: shell
('10.0.0.2', 10802) 514/tcp cmd
Timeout ...
Timeout ...

Key: rootd
('10.0.0.4', 10804) 1094/tcp
('10.0.0.4', 10804) 1094/udp
Timeout ...
Timeout ...

Key: mysql
('10.0.0.4', 10804) 3306/tcp
('10.0.0.4', 10804) 3306/udp
Timeout ...
Timeout ...

Key: █
```

## 7.2 Servent 1

```
root@mininet-vm:~/Peer-to-Peer-Storing-System/src# python servent.py 10802 ../tst/pairs1.txt 10.0.0.3:10803
Starting...
Running at: 10.0.0.2 10802
Processing package from: ('10.0.0.1', 58012)
Answering client ('10.0.0.1', 58012) ...
Searching for: fido ...

Processing package from: ('10.0.0.1', 58012)
Answering client ('10.0.0.1', 58012) ...
Searching for: tfido ...

Processing package from: ('10.0.0.1', 58012)
Answering client ('10.0.0.1', 58012) ...
Searching for: shell ...
Found!

Processing package from: ('10.0.0.1', 58012)
Answering client ('10.0.0.1', 58012) ...
Searching for: rootd ...

Processing package from: ('10.0.0.1', 58012)
Answering client ('10.0.0.1', 58012) ...
Searching for: mysql ...
```

## 7.3 Servent 2

```
root@mininet-vm:~/Peer-to-Peer-Storing-System/src# python servent.py 10803 ../tst/pairs2.txt 10.0.0.4:10804
Starting...
Running at: 10.0.0.3 10803
Processing package from: ('10.0.0.2', 10802)
Answering client ('10.0.0.1', 58012) ...
Searching for: fido ...

Processing package from: ('10.0.0.2', 10802)
Answering client ('10.0.0.1', 58012) ...
Searching for: tfido ...

Processing package from: ('10.0.0.2', 10802)
Answering client ('10.0.0.1', 58012) ...
Searching for: shell ...

Processing package from: ('10.0.0.2', 10802)
Answering client ('10.0.0.1', 58012) ...
Searching for: rootd ...

Processing package from: ('10.0.0.2', 10802)
Answering client ('10.0.0.1', 58012) ...
Searching for: mysql ...
```

## 7.4 Servent 3

```
root@mininet-vms:~/Peer-to-Peer-Storing-System/src# python servent.py 10804 ../tst/pairs3.txt 10.0.0.5:10805
Starting...
Running at: 10.0.0.4 10804
Processing package from: ('10.0.0.3', 10803)
Answering client ('10.0.0.1', 58012) ...
Searching for: fido ...

Processing package from: ('10.0.0.3', 10803)
Answering client ('10.0.0.1', 58012) ...
Searching for: tfido ...

Processing package from: ('10.0.0.3', 10803)
Answering client ('10.0.0.1', 58012) ...
Searching for: shell ...

Processing package from: ('10.0.0.3', 10803)
Answering client ('10.0.0.1', 58012) ...
Searching for: rootd ...
Found!

Processing package from: ('10.0.0.3', 10803)
Answering client ('10.0.0.1', 58012) ...
Searching for: mysql ...
Found!
```

## 7.5 Servent 4

```
root@mininet-vms:~/Peer-to-Peer-Storing-System/src# python servent.py 10805 ../tst/pairs4.txt 10.0.0.6:10806
Starting...
Running at: 10.0.0.5 10805
Processing package from: ('10.0.0.4', 10804)
Answering client ('10.0.0.1', 58012) ...
Searching for: fido ...

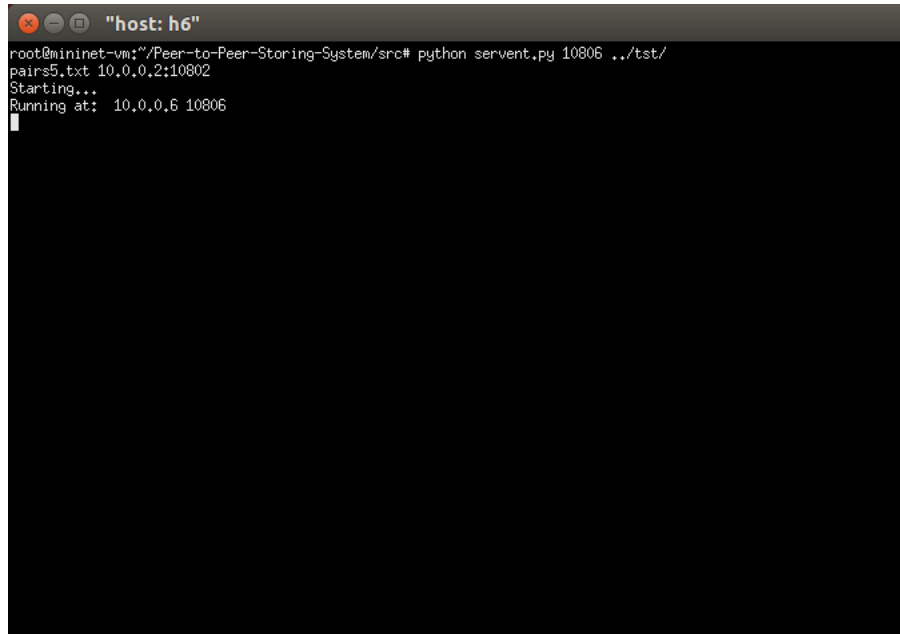
Processing package from: ('10.0.0.4', 10804)
Answering client ('10.0.0.1', 58012) ...
Searching for: tfido ...

Processing package from: ('10.0.0.4', 10804)
Answering client ('10.0.0.1', 58012) ...
Searching for: shell ...

Processing package from: ('10.0.0.4', 10804)
Answering client ('10.0.0.1', 58012) ...
Searching for: rootd ...

Processing package from: ('10.0.0.4', 10804)
Answering client ('10.0.0.1', 58012) ...
Searching for: mysql ...
```

## 7.6 Servent 5



```
root@mininet-vm:~/Peer-to-Peer-Storing-System/src# python servent.py 10806 ../tst/
pairs5.txt 10.0.0.2:10802
Starting...
Running at: 10.0.0.6 10806
```

## 8 Conclusão

Com esse trabalho também foi possível ver na prática conceitos ministrados em sala de aula, fixando o conteúdo. Foi interessante ver como se comportam redes sobrepostas e o alagamento confiável, que apesar de ser um protocolo simples, tem uma implementação muito interessante. Nossa maior dificuldade foi na implementação dos servents, para mantê-los sincronizados, tentando evitar envio descontrolado de mensagens e loops infinitos.