

# Comunicações Por Computador

## 2017/2018

Carlos Pedrosa<sup>1</sup>, David Sousa<sup>1</sup>, and Manuel Sousa<sup>1</sup>

Universidade Do Minho  
a{77320,78938,78869}@alunos.uminho.pt

**Abstract.** Este relatório aborta todas as etapas realizadas pelo grupo para a implementação de um ReverseProxy.

**Keywords:** ReverseProxy · UDP · Agente · Monitor.

## 1 Introdução

Este trabalho tem como objectivo o desenho e a implementação de um servidor ReverseProxy. De facto, um servidor deste tipo tem como objectivo lidar com as diversas conexões que recebe e redireccioná-las para o servidor mais adequado para lidar com o pedido. Este tipo de servidor é principalmente usado para distribuir a carga entre os vários servidores back-end. Assim, neste documento iremos detalhar todo o processo usado para concluir o desafio com sucesso.

## 2 Arquitetura da Solução

Desde o início do projecto que a solução sempre passou por duas fases. A primeira consiste principalmente em encontrar servidores back-end, neste caso em particular, servidores HTTP. Este processo é feito com base em UDP (User Datagram Protocol). Existirá um componente do projecto responsável por enviar pacotes UDP em multicast. Os servidores HTTP irão correr em paralelo um componente capaz de receber estes pacotes e enviar em unicast dados para que o ReverseProxy possa atualizar a sua tabela de servidores. A segunda fase do projecto consiste em receber conexões TCP na porta 80 do ReverseProxy, calcular o melhor servidor para receber aquela conexão e abrir uma conexão para esse mesmo servidor. O processo desde o pedido de um cliente até a resposta é simples. Quando um cliente envia um pedido de conexão ao ReverseProxy, este aceita-a e cria uma Thread Worker para ligar com esta. A Thread Worker criada, calcula o servidor HTTP para se conectar, estabelece a conexão e cria uma Thread ReceiverWorker que irá receber os dados vindos do servidor HTTP. Todos os dados que a Thread Worker recebe vindos do cliente são passados para o servidor HTTP. O servidor HTTP interpreta o pedido e envia os dados de resultado para a Thread ReceiverWorker, que os redireciona para o Cliente. De notar que no decorrer de toda esta lógica, existe em segundo plano toda uma troca de pacotes UDP para atualizar a tabela de servidores. Toda esta lógica UDP será abordada num capítulo mais adiante.

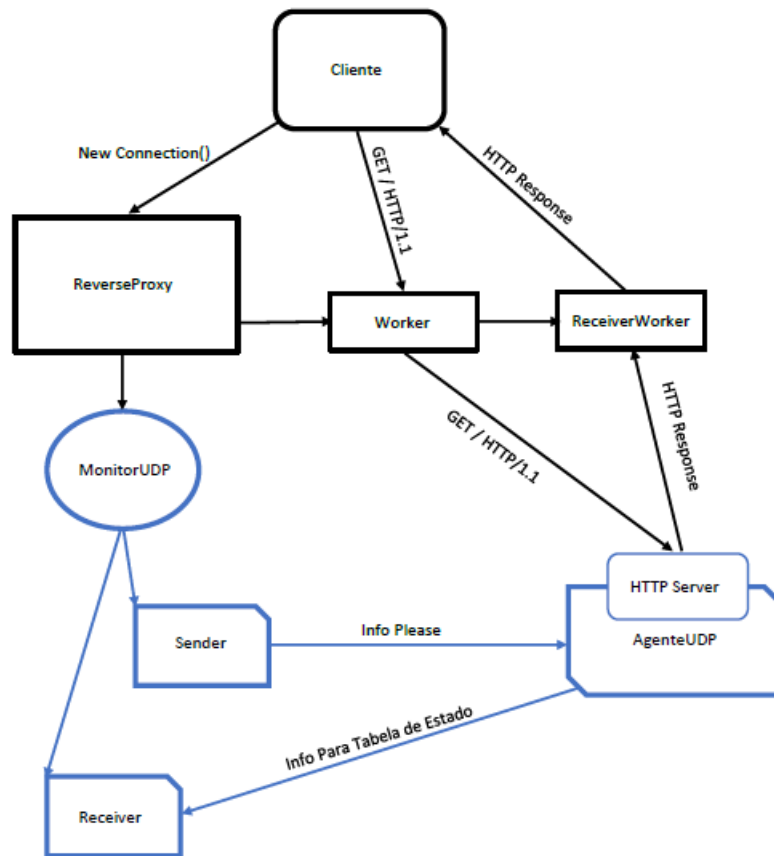


Fig. 1. Arquitetura do Projecto com um Servidor/Agente.

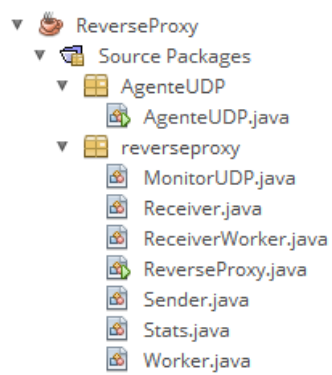


Fig. 2. Divisão do projecto em JAVA.

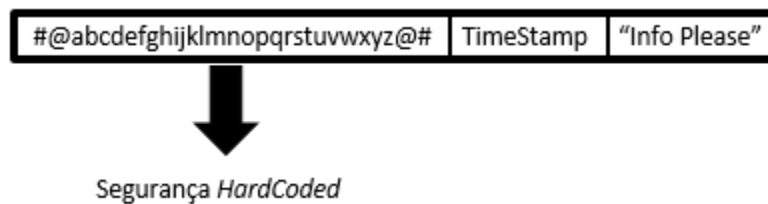
## 2.1 Especificação do Protocolo UDP

Nesta secção iremos especificar toda a estrutura UDP do projecto. Antes do ReverseProxy começar a aceitar conexões é iniciada uma Thread(MonitorUDP) que irá tratar de todo o envio e recolha de pacotes UDP. Este MonitorUDP irá por sua vez criar duas Threads auxiliares. Uma capaz de enviar pacotes UDP e outra capaz de receber os pacotes enviados pelo AgenteUDP (processo que, em paralelo, corre nos servidores HTTP).

### 2.1.1 Formato das mensagens UDP

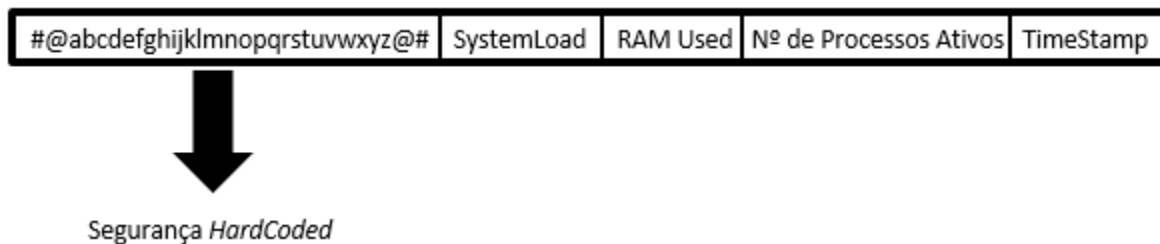
Como mencionado anteriormente existem dois tipos de pacotes UDP a circular, um enviado pelo MonitorUDP, em multicast, para os AgentesUDP, de modo a identificarem novos servidores HTTP, e um que viaja no sentido inverso, isto é, dos AgentesUDP para o MonitorUDP e que transporta a informação necessária para o MonitorUDP atualizar a sua tabela de servidores.

O pacote UDP enviado pelo MonitorUDP é relativamente simples. Trata-se apenas de uma string de segurança seguido do timestamp de envio e com uma pequena string a requisitar informação.



**Fig. 3.** Estrutura do pacote UDP enviado pelo MonitorUDP.

Por outro lado, o pacote UDP enviado pelo AgenteUDP é um pouco mais complexo. Este contém também uma string de segurança, um conjunto de medições efetuadas pelo AgenteUDP no servidor HTTP, como a carga de CPU, a RAM usada, o número de processos ativos e, por fim, o timestamp do pacote UDP recebido para calcular o RTT no MonitorUDP.



**Fig. 4.** Estrutura do pacote UDP enviado pelo AgenteUDP.

### 2.1.2 Interações entre Componentes UDP

Neste projecto, existem 2 tipos de interações UDP: pedidos e respostas.

O monitorUDP(Thread Sender) entre cada 0 a 10 segundos envia um pacote UDP em multicast para pedir informações sobre os servidores HTTP. Estes servidores, devidamente equipados com um agenteUDP capaz de receber e interpretar esses pacotes, recolhem informação do servidor e enviam-na de volta para o MonitorUDP(Thread Receiver) em unicast. Podemos perceber melhor estas interações com o esquema abaixo.

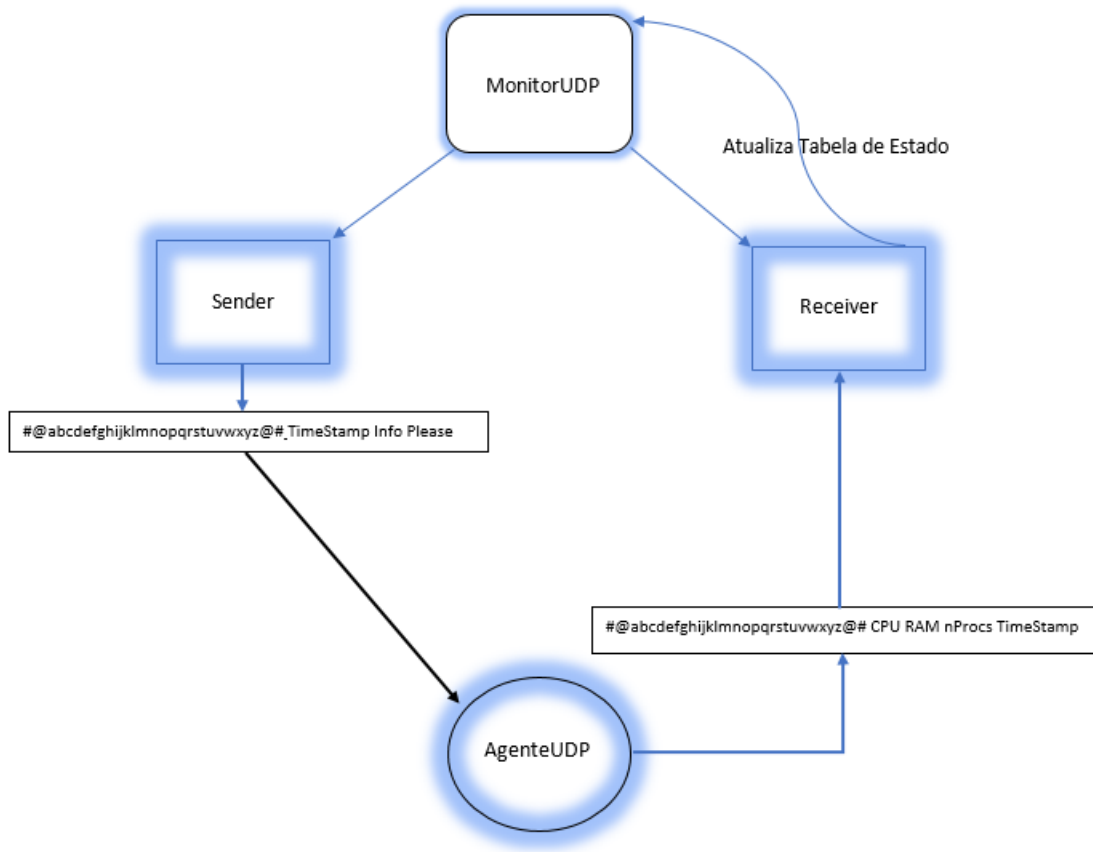


Fig. 5. Estrutura do pacote UDP enviado pelo AgenteUDP.

## 2.2 Implementação

A implementação deste projecto foi na linguagem de programação JAVA. Esta é uma linguagem bastante versátil e portanto contém todos os pacotes necessários à implementação do ReverseProxy. Para lidar com UDP usamos os seguintes pacotes:

```
import java.net.InetAddress;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.MulticastSocket;
```

Segue-se uma breve descrição de cada classe do projecto.

### 2.2.1. ReverseProxy

Classe principal do projecto. Contém a tabela de estado, estrutura que irá conter todos os servidores HTTP e as suas informações. Quando o programa inicia, é criada um Thread MonitorUDP para lidar com o tráfego de pacotes UDP. De seguida, o servidor começa a aceitar conexões de clientes na porta 80. Quando aceita uma chama uma Thread Worker para lidar com essa conexão.

### 2.2.2. MonitorUDP

Quando iniciada, lança duas Threads(Sender e Receiver) para lidar com o UDP. De 15 em 15 segundos, verifica se existe algum servidor que não respondeu a 3 pedidos consecutivos. Caso exista, remove-o da tabela de estado.

### 2.2.3. Sender

Cria o pacote UDP a enviar em multicast, e envia-o de x em x segundos ( $x = 0, \dots, 10$ ) para o endereço multicast.

### 2.2.4. Receiver

Recebe os pacotes UDP enviados pelos diversos AgenteUDP em unicast, verifica a fiabilidade do pacote através da segurança hardcoded, e extrai as informações necessárias para atualizar a tabela de estado.

### 2.2.5. Worker

Responsável por lidar com as conexões estabelecidas pelos clientes. Determina o servidor HTTP a estabelecer conexão, cria um socket TCP para esse servidor, invoca uma Thread ReceiverWorker e tudo o que recebe do cliente redireciona para o servidor HTTP.

### 2.2.6. ReceiverWorker

Lida com a informação recebida do servidor HTTP, redirecionando-a para o cliente.

### 2.2.7. Stats

Classe que contém as informações a guardar de cada servidor.

```
public class Stats {
    private String ip;           // IP do servidor HTTP;
    private int port;           // Porta do servidor;
    private int n;               // Numero de Vezes Seguidas
    que o Servidor n respondeu ao Pedido UDP;
    private double cpuUsed;      // CPU usado pelo servidor;
    private double ramUsed;      // Ram usada pelo servidor;
    private int nProc;           // Numero de Processos ativos no servidor;
    private double rtt;          // Round Trip Time;
    private double bW;           // Largura de Banda Usado Pelo Servidor;
}
```

### 2.2.8. AgenteUDP

Classe principal que irá correr em paralelo com o servidor HTTP. Recebe pedidos UDP, enviados pelo Sender, recolhe informação e envia-a em unicast para o Receiver.

### 2.2.9. Algoritmo de Seleção do Servidor

Para escolher o servidor a qual conectar pedidos de clientes foi necessário estipular um algoritmo de seleção. Optamos por utilizar um algoritmo baseado numa média pesada. Assim, o algoritmo escolhido atribui pesos de 50% à carga no sistema, 40% à RAM usada, 5% ao número de processos a correr pois grande número de processos não implica necessariamente alto uso de CPU nem de RAM e 5% ao RTT(Round Trip Time).

```
public double getLoad() {
    double load = 0.5 * this.cpuUsed + 0.4 * this.ramUsed + 0.05 * this.nProc + 0.05 * this.rtt;

    return load;
}
```

## 2.3 Testes e Resultados

Como maneira de testar o projecto desenvolvido decidimos usar a plataforma CORE. Assim, desenhamos a seguinte topologia com 2 Servidores HTTP, um ReverseProxy e um Cliente.

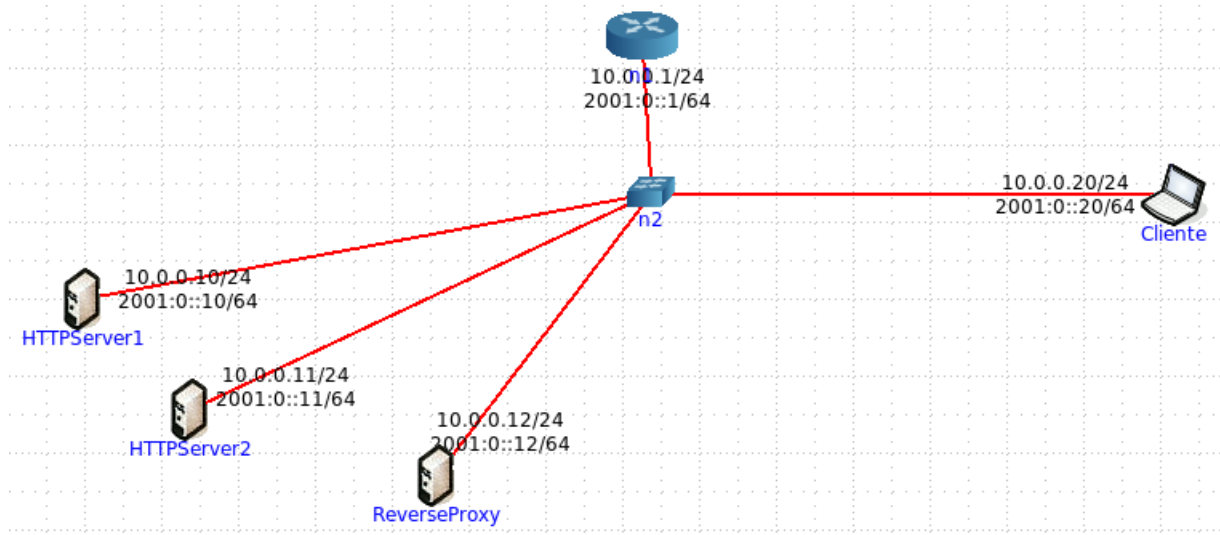
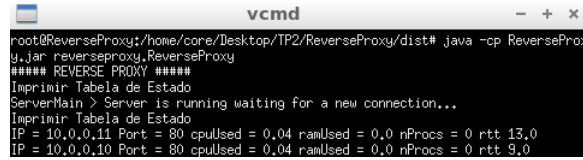


Fig. 6. Topologia usada para teste no CORE.

Colocou-se um servidor HTTP a correr em cada um dos HTTPServer e iniciou-se também em cada um deles o AgenteUDP. Passados 15 segundos, e apesar do teste não ter sido efetuado na perfeição devido a problemas técnicos, é possível observar a tabela de estado.



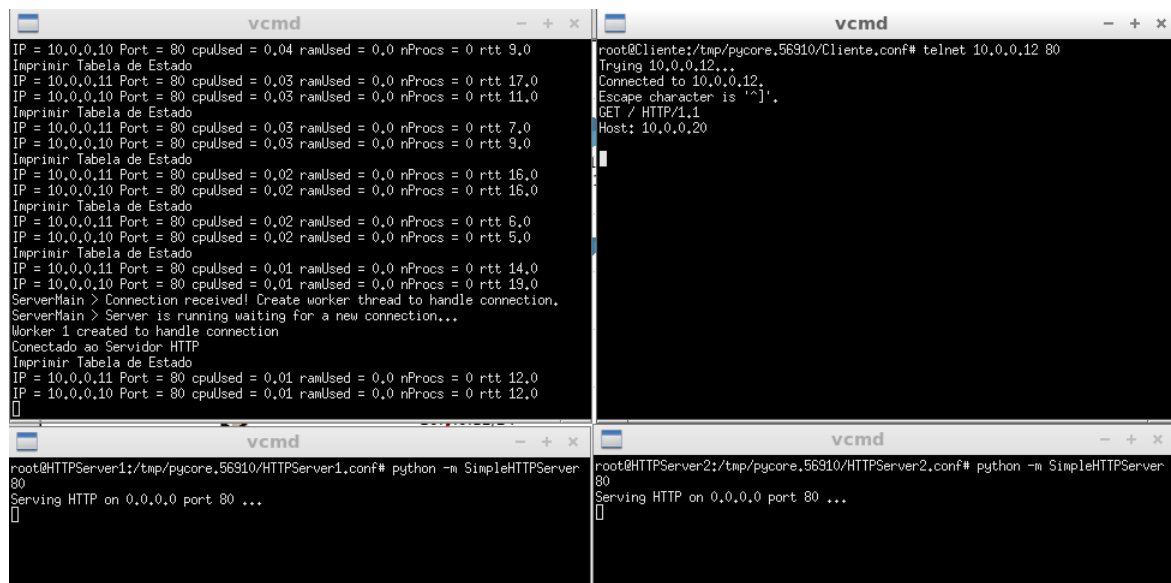
```

root@ReverseProxy:/home/core/Desktop/TP2/ReverseProxy/dist# java -cp ReverseProx
y.jar reverseproxy.ReverseProxy
##### REVERSE PROXY #####
Imprimir Tabela de Estado
ServerMain > Server is running waiting for a new connection...
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0.04 ramUsed = 0.0 nProcs = 0 rtt 13.0
IP = 10.0.0.10 Port = 80 cpuUsed = 0.04 ramUsed = 0.0 nProcs = 0 rtt 9.0

```

Fig. 7. Tabela de Estado.

O próximo passo é a conexão de um cliente ao ReverseProxy. Podemos observar que esta foi efetuada com sucesso. Segue-se o pedido HTTP do cliente.



```

root@ReverseProxy:/home/core/Desktop/TP2/ReverseProxy/dist# java -cp ReverseProx
y.jar reverseproxy.ReverseProxy
##### REVERSE PROXY #####
Imprimir Tabela de Estado
ServerMain > Server is running waiting for a new connection...
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0.04 ramUsed = 0.0 nProcs = 0 rtt 13.0
IP = 10.0.0.10 Port = 80 cpuUsed = 0.04 ramUsed = 0.0 nProcs = 0 rtt 9.0

root@Cliente:/tmp/pycore.56910/Cliente.conf# telnet 10.0.0.12 80
Trying 10.0.0.12...
Connected to 10.0.0.12.
Escape character is '^]'.
GET / HTTP/1.1
Host: 10.0.0.20

root@HTTPServer1:/tmp/pycore.56910/HTTPServer1.conf# python -m SimpleHTTPServer
80
Serving HTTP on 0.0.0.0 port 80 ...

root@HTTPServer2:/tmp/pycore.56910/HTTPServer2.conf# python -m SimpleHTTPServer
80
Serving HTTP on 0.0.0.0 port 80 ...

```

Fig. 8. Teste de conexão ao ReverseProxy.

Por fim, era esperar que o servidor devolve-se a resposta ao pedido HTTP. Podemos observar que tal acontece e que foi o servidorHTTP1 a dar resposta ao pedido pelo HTML devolvido e pelo LOG no servidor HTTP.

The figure displays four terminal windows. The top-left window shows a loop of printing system statistics (IP, Port, CPU, RAM, Procs, RTT) for various IP addresses. The top-right window shows a telnet session where a client connects to 10.0.0.12 on port 80 and sends a GET request. The bottom-left window shows an HTTP server (SimpleHTTPServer) running on 0.0.0.0 port 80, receiving the GET request. The bottom-right window shows another instance of the SimpleHTTPServer running on 0.0.0.0 port 80.

```

vcmd
IP = 10.0.0.10 Port = 80 cpuUsed = 0,03 ramUsed = 0,0 nProcs = 0 rtt 9,0
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,02 ramUsed = 0,0 nProcs = 0 rtt 16,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,02 ramUsed = 0,0 nProcs = 0 rtt 16,0
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,02 ramUsed = 0,0 nProcs = 0 rtt 6,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,02 ramUsed = 0,0 nProcs = 0 rtt 5,0
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 14,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 19,0
ServerMain > Connection received! Create worker thread to handle connection.
ServerMain > Server is running waiting for a new connection...
Worker 1 created to handle connection
Conectado ao Servidor HTTP
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 12,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 12,0
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 17,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 6,0
Imprimir Tabela de Estado
IP = 10.0.0.11 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 15,0
IP = 10.0.0.10 Port = 80 cpuUsed = 0,01 ramUsed = 0,0 nProcs = 0 rtt 18,0
[]

vcmd
root@Cliente:/tmp/pycore.56910/Cliente.conf# telnet 10.0.0.12 80
Trying 10.0.0.12...
Connected to 10.0.0.12.
Escape character is '^['.
GET / HTTP/1.1
Host: 10.0.0.20

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.3
Date: Wed, 23 May 2018 17:52:45 GMT
Content-type: text/html
Content-Length: 60
Last-Modified: Wed, 23 May 2018 17:46:31 GMT

<html>
<head>
CC - TP2 - Server1
</head>
</html>
[]

vcmd
root@HTTPServer1:/tmp/pycore.56910/HTTPServer1.conf# python -m SimpleHTTPServer
80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.12 - - [23/May/2018 10:52:45] "GET / HTTP/1.1" 200 -
[]

vcmd
root@HTTPServer2:/tmp/pycore.56910/HTTPServer2.conf# python -m SimpleHTTPServer
80
Serving HTTP on 0.0.0.0 port 80 ...
[]

```

Fig. 9. Resposta Obtida Pelo Cliente.

## References

1. MulticastSocket, <https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html>, 26-03-2018.
2. DatagramSocket, <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html>, 27-03-2018.
3. MulticastSocket, <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html>, 27-03-2018.