

# Proxy TCP reverse com monitorização proactiva

Marques, Ricardo Jorge<sup>1</sup>

<sup>1</sup> Universidade do Minho, Largo do Paço, 4704-553 Braga, Portugal  
a55723@alunos.uminho.pt

**Abstract.** Este relatório foca-se no desenvolvimento de um reverse proxy HTTP transparente a funcionar sobre o protocolo TCP e monitorizado através de um protocolo sobre o protocolo UDP. Foca-se sobretudo nos aspetos de balanceamento de carga e implementa um método de escolha do melhor servidor para dar resposta ao pedido de um cliente.

**Keywords:** Proxy, TCP, UDP, HTTP, Load Balance.

## 1 Arquitetura da Solução implementada

A solução implementada apresenta-se sobre a forma de duas aplicações independentes, desenvolvidas com base na tecnologia JAVA 8. A primeira aplicação tem a função de monitorizar os servidores HTTP e informar o proxy de que estão ativos. A segunda aplicação tem a função de manter uma lista atualizada com informação dos servidores HTTP e atuar como ponto de interligação inteligente, utilizando um algoritmo de escolha do melhor servidor a cada novo pedido do cliente.

### 1.1 Monitor – “ClienteUDP”

A aplicação ClienteUDP é implementada em dois processos distintos, que correm de forma isolada.

A primeira função que esta aplicação tem é a de informar o proxy de que o servidor HTTP se encontra ativo e pronto a receber pedidos. Nesse sentido este primeiro processo consiste em informar, em intervalos periódicos fixos, o proxy, através de um datagrama UDP.

A segunda função consiste em responder aos pedidos de informação enviados pelo proxy. Este segundo processo, consiste em estar à escuta de uma porta pré-definida, no protocolo UDP e responder aos datagramas de pedido de informação vindos do proxy.

### 1.2 Servidor Proxy – “Servidor”

A aplicação Servidor é implementada sobre a forma de uma aplicação “Multithread” que faz recurso à utilização de estruturas de dados partilhados.

Esta aplicação divide-se em dois grandes grupos, que trabalham interligados. O primeiro grande grupo foca-se na monitorização dos servidores HTTP, armazenando em estruturas de dados adequadas um histórico de cada servidor HTTP. Este histórico serve

para calcular o servidor mais adequado para o proxy redirecionar os pedidos dos clientes.

O cálculo do melhor servidor é conseguido através da medição de três métricas distintas: Round Trip Time, número de ligações ativas no servidor HTTP e a percentagem de perdas de pacotes no link, sendo esta percentagem dividida em duas métricas: a perda total no link desde o estabelecimento da conexão e a perda nos últimos 20 pedidos.

A fórmula de cálculo assenta no seguinte polinómio:

$$f(x,y,z) = (a * ((x1 + x2) / 2) + by + cz) \quad (1)$$

Sendo  $x1$ ,  $x2$ ,  $y$  e  $z$ , a perda total no link, a perda nos últimos 20 datagramas, o Round Trip Time e o numero de ligações, respetivamente. Todos estes valores são obtidos como percentual do valor máximo encontrado nas métricas de todos os servidores.

Os valores de  $a$ ,  $b$  e  $c$  são os fatores de peso de cada uma das métricas no polinómio, que são totalmente configuráveis. A soma destes fatores deve ser igual a 1.

O resultado é expresso sob a forma de percentagem, onde o melhor servidor é aquele que apresenta a percentagem de carga mais baixa.

O facto de existirem duas métricas para a percentagem de perdas é explicado pela necessidade de tornar o algoritmo mais reativo às perdas momentâneas no link. Como a percentagem total de perdas no link é uma medida com uma amostragem longa, existe a necessidade de ajustar as perdas com uma medida mais curta, evitando que o ajuste lento da percentagem de perdas total possa pesar de forma errada no algoritmo. Assim, monitoriza-se o estado dos últimos 20 pedidos e corrige-se o valor da medida longa, fazendo uma média aritmética destas duas métricas.

O segundo grupo foca-se na interligação entre cliente e servidor, aceitando pedidos HTTP de um ou mais clientes e redirecionando os pedidos para o servidor mais adequado a cada instante. Esta função de proxy é totalmente transparente, não guardando qualquer tipo de informação sobre o cliente. Aceita múltiplas conexões de um ou vários clientes e funciona em HTTP0.9, HTTP1.0 e HTTP1.1.

## 2 Especificação do protocolo de monitorização

O protocolo de monitorização dos servidores HTTP consiste em duas operações distintas. A primeira operação consiste no envio periódico de um datagrama UDP no sentido do monitor para o proxy. A segunda operação consiste no envio periódico de um datagrama UDP no sentido do proxy para o monitor, ao qual o monitor responde com um datagrama UDP contendo informação que vai atualizar as estruturas de dados que registam o histórico de cada servidor HTTP.

### 2.1 Primitivas de comunicação

Para este projeto foram utilizadas as seguintes primitivas de comunicação do JAVA:

**receive(DatagramPacket p).** Função que lê de um socket um datagrama UDP.

**send(DatagramPacket p).** Função que envia por um socket um datagrama UDP.

**int read(byte[] b, int off, int len).** Função que lê de uma inputStream um conjunto de bytes a partir do offset *off* e de tamanho *len* e armazena o resultado no array *b*. Retorna o número de bytes lidos

**void write(byte[] b).** Função que escreve numa outputStream o array de bytes *b*.

## 2.2 Formato das mensagens Protocolares

As mensagens protocolares são Strings em puro texto, de tamanho variável e onde os campos de dados estão delimitados por espaços. São mensagens protocolares os seguintes exemplos:

**Status.** Mensagem enviada no sentido do monitor para o proxy, que informa o proxy de que o servidor HTTP está disponível a aceitar ligações e que apresenta o seguinte formato:

“S 1111”

Sendo ‘S’ o caracter que indica o tipo de mensagem.

Sendo 1111 uma representação de um número inteiro que é acumulado a cada nova mensagem enviada.

**Information.** Mensagem periódica no sentido do proxy para o monitor, que pede ao monitor o estado em que este se encontra e que apresenta o seguinte formato:

“I 1111 22222222”

Sendo ‘I’ o caracter que indica o tipo de mensagem

Sendo 1111 uma representação de um número inteiro que é acumulado a cada nova mensagem enviada.

Sendo 2222 uma representação de um número longo que representa um timestamp do momento em que a mensagem foi enviada no proxy.

**Replay.** Mensagem de resposta ao pedido Information, que vai no sentido do monitor para o proxy. Apresenta o seguinte formato:

“R 1111 22222222 3333”

Sendo ‘R’ o caracter que indica o tipo de mensagem

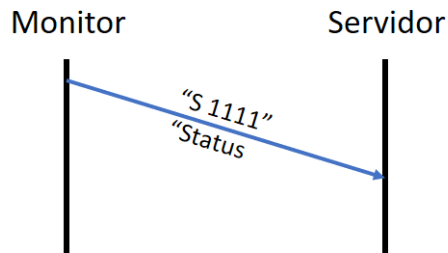
Sendo 1111 uma representação de um número inteiro que é acumulado a cada nova mensagem enviada, e que vem no pacote Information.

Sendo 2222 uma representação de um número longo que representa um timestamp do momento em que a mensagem foi enviada no proxy, e que vem no pacote Information.

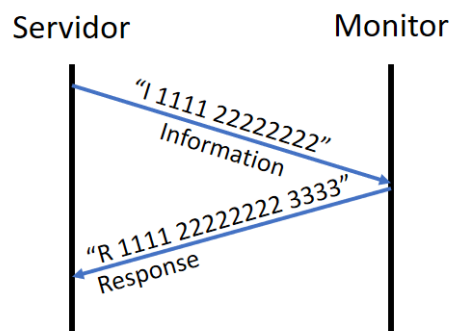
Sendo 3333 o número de ligações TCP ativas que o servidor HTTP tem no momento.

### 2.3 Interações

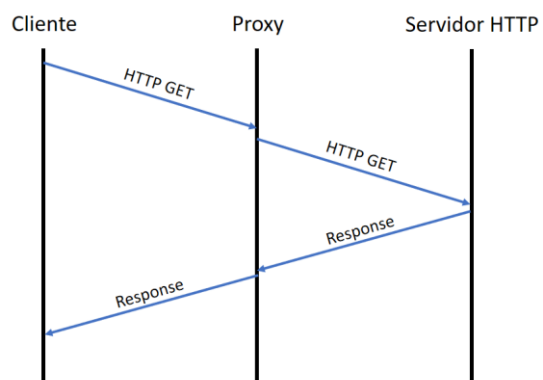
Existem, de momento apenas três tipos de interações. As mesmas podem ser vistas nos seguintes diagramas de sequência.



**Fig. 1.** Envio de status do monitor para o proxy através do protocolo UDP



**Fig. 2.** Pedido de informação do Servidor para o Monitor e respetiva resposta, através do protocolo UDP



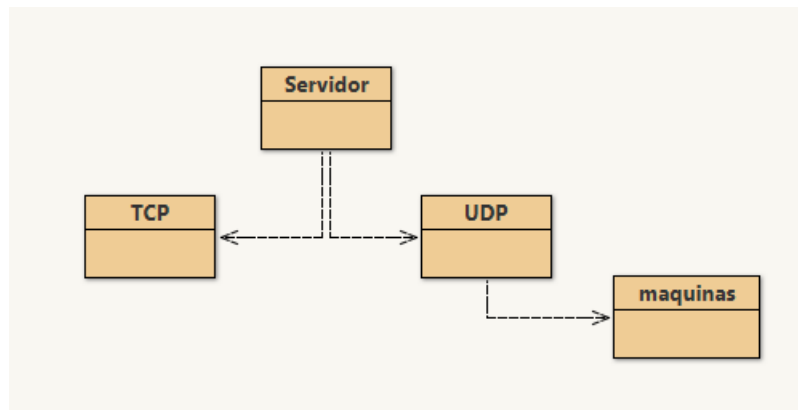
**Fig. 3.** Pedido de uma página de internet do Cliente ao proxy através do protocolo TCP

### 3 Implementação

A implementação deste projeto foi feita com base na tecnologia JAVA 8. A seguir mostram-se os diagramas de classes para cada uma das aplicações, Monitor e servidor e posteriormente explica-se cada uma das classes desenvolvidas.



**Fig. 4.** Diagrama de Classes da aplicação ClienteUDP. (monitor)



**Fig. 5.** Diagrama de Classes da aplicação Servidor. (proxy)

#### 3.1 Aplicação ClienteUDP

A aplicação ClienteUDP é uma aplicação que tem apenas uma classe. Fazem parte da aplicação ClienteUDP os seguintes atributos, métodos e ficheiros de configuração.

**Table 1.** Exemplo de um ficheiro de configuração da aplicação ClienteUDP

<u>Cliente.conf</u>
# endereco do proxy
endereco=10.0.0.13
# porta UDP do proxy
porta=5555
#frequencia em ms com que vai fazer pooling ao proxy
frequencia=3000

**Table 2.** Atributos da classe ClienteUDP

Atributo	Função
private static Integer acumulador	Variável iniciada com o valor 1 e que é incrementada a cada envio de status do monitor.
private static int porta	Variável que contém o número da porta para estabelecimento do socket entre o monitor e o servidor proxy
private static int freqAlive	Variável que contém o valor em milissegundos para o envio do datagrama de status do monitor
private static String endereço	Variável que contém o endereço do servidor proxy

**Table 3.** Métodos da classe ClienteUDP

Método	Função
	Método principal. Recebe como argumentos o nome de um ficheiro de configuração.
public static void main(String[] args)	Trata de estabelecer um processo para o envio do datagrama de status ao servidor Proxy. Trata também de ficar à escuta de uma porta UDP e responder aos pedidos do servidor proxy.
private static void readConfig(String ficheiro) throws IOException	Método responsável pela leitura e parse do ficheiro de configuração.
private static void send(String host, int port, byte[] message)	Método responsável por enviar datagramas UDP para o host e porta passados como parâmetros.
private static void sendAlive()	Método que constrói e envia o datagrama de status para o servidor proxy.
public static Integer netStatus()	Método que, através de uma chamada ao sistema do comando netstat, retorna o número de ligações TCP ativas.

### 3.2 Aplicação Servidor

A aplicação servidor está dividida em 4 classes e um ficheiro de configuração. A seguir apresentamos em exemplo de um ficheiro de configuração e a descrição de cada classe detalhadamente.

**Table 4.** Exemplo de um ficheiro de configuração da aplicação Servidor

<u>Servidor.conf</u>
# porta UDP do proxy
portaUDP=5555
# porta TCP do proxu
portaTCP=80
# tempo em ms de repetição de pedido de informação aos servidores http

```

probeTIMER=1000
# tempo em ms de repetição de verificação de timeout dos servidores http
probeTimeout=10000
# tempo em ms para o timeout
timeout=15000
# coeficientes de calculo para escolha do melhor servidor
# a soma dos tres coeficientes deve ser igual a 1
pesoPerdas=0.5
pesoRTT=0.25
pesoLigacoes=0.25

```

---

**Servidor.** A classe Servidor é a classe principal da aplicação Servidor. Dela fazem parte os seguintes atributos e método.

**Table 5.** Atributos da classe Servidor

Atributo	Função
private static int udpPort	Variável que contem o número da porta usada para o protocolo UDP
private static int tcpPort	Variável que contem o número da porta usada para o protocolo TCP
private static long probeTimer	Variável que contém o valor em milissegundos para o envio do datagrama information.
private static long timeoutTimer	Variável que contém o valor em milissegundos para execução do teste de verificação de que os servidores HTTP estão online
private static long timeoutDelay	Variável que contém o valor em milissegundos para o máximo de timeout dos servidores HTTP
private static float pesoPerdas	Variável que contém o valor do peso das perdas na formula de calculo do melhor servidor
private static float pesoRTT	Variável que contém o valor do Round Trip Time na formula de calculo do melhor servidor
private static float pesoLigacoes	Variável que contém o valor do peso do numero de ligações na formula de calculo do melhor servidor

**Table 6.** Métodos da classe Servidor

Método	Função
public static void main(String[] args) throws InterruptedException, IOException	Método principal. Recebe como argumentos o nome de um ficheiro de configuração. É também responsável por ficar á escuta da porta TCP e aceitar pedidos dos clientes. E por arran- car o processo da classe UDP

<code>private static void readConfig(String fi- cheiro) throws IOException</code>	Método responsável pela leitura e parse do fi- cheiro de configuração.
---	---

**UDP.** A classe UDP estende a classe nativa Thread. Esta classe é responsável por gerir, recolher e armazenar informação dos servidores http, comunicando com os monitores. É responsável por calcular o melhor servidor e garantir que os servidores que atingiram timeout são removidos da tabela de servidores ativos. Dela fazem parte os seguintes atributos e métodos.

**Table 7.** Atributos da classe UDP

Atributo	Função
<code>private static Map&lt;String, maqui- nas&gt; pollingTable</code>	Map que relaciona o endereço de cada máquina com a sua informação.
<code>private static Map&lt;String, Long&gt; statusTable</code>	Map que relaciona o endereço de cada máquina com um timestamp da última vez que o proxy recebeu um data-grama de status.
<code>private static ScheduledExecutor- Service MaquinasExec;</code>	Pool do serviço de execuções.
<code>private static ScheduledFuture&lt;?&gt; testMachines;</code>	Processo de teste de timeout dos servidores HTTP.
<code>private static ScheduledFuture&lt;?&gt; probeMachines;</code>	Processo que envia pedido de status aos servidores HTTP.
<code>private Thread t;</code>	Thread onde corre a função o método principal da classe UDP
<code>private static boolean status;</code>	Variável responsável por manter ou terminar o processo de escuta do socket UDP.
<code>private static int porta;</code>	Variável que contém o numero de porta usado no protocolo UDP.
<code>private static long delayProbe;</code>	Variável que contém o tempo em milissegundos para a repetição do pedido de status aos servidores HTTP.
<code>private static long delayTest;</code>	Variável que contém o tempo em milissegundos para executar o teste de timeout dos servidores HTTP.
<code>private static long timeout;</code>	Variável que contém o valor máximo, em milissegundos, do timeout dos servidores HTTP.
<code>private long maxRTT;</code>	Variável que contém o valor do máximo do Round Trip Time de entre todos os servidores HTTP.
<code>private long maxLIG;</code>	Variável que contém o valor do máximo do numero de ligações de entre todos os servidores HTTP.
<code>private float maxLOST;</code>	Variável que contém o valor do máximo da percentagem total de pacotes perdidos de entre todos os servidores HTTP.



<code>private float maxSLOST;</code>	Variável que contém o valor do máximo da percentagem de pacotes perdidos dos ultimo 20 pedidos efetuados de entre todos os servidores HTTP.
<code>private static String adr;</code>	Variável que contém o endereço do servidor com melhor pontuação.
<code>private static float pontuacao;</code>	Variável que contem o valor da melhor pontuação de entre todos os servidores.
<code>private static float pesoPerdas;</code>	Variável que contem o valor do peso das perdas na formula de escolha do melhor servidor
<code>private static float pesoRTT;</code>	Variável que contem o valor do peso do Round Trip Time na formula de escolha do melhor servidor
<code>private static float pesoLigacoes;</code>	Variável que contem o valor do peso do numero de ligações na formula de escolha do melhor servidor

**Table 8.** Métodos da classe UDP

Método	Função
<code>UDP(int prt, long dp, long dt, long tOut, float pp, float pr, float pl)</code>	Método que inicia uma nova instancia da classe UDP
<code>public void run()</code>	Método principal da classe UDP. É responsável por agendar os pedidos periódicos de informação e a verificação dos timeouts dos servidores HTTP.
<code>public void start()</code>	Método que inicia a uma thread onde executa o método <code>run()</code> .
<code>public void stopUDP()</code>	Método que para a thread iniciada pelo método <code>start()</code> .
<code>public String getBestServer()</code>	Método que calcula, de entre todos os servidores, o mais disponível para reenviar os pedidos dos clientes.
<code>private static void startUDP()</code>	Método que fica à escuta da porta UDP e redireciona os pedidos para a função de parse de cada mensagem
<code>private static void parseMensagem(String mensagem, String addr, long time)</code>	Método que verifica cada datagrama UDP e atualiza as tabelas dos servidores HTTP com nova informação
<code>private static void testMaquinas(long delay)</code>	Método que verifica se os servidores não respondem à mais que x milissegundos e em caso afirmativo os remove da tabela de servidores ativos.
<code>private static void probeMaquinas()</code>	Método que, para cada servidor da tabela de servidores ativos, envia o pedido de informação ao servidor HTTP.
<code>private static void send(String host, int port, byte[] message)</code>	Método que envia PDU's através de UDP para um servidor HTTP.

**TCP.** A classe TCP estende a classe nativa Thread. Esta classe é responsável por interligar os clientes aos servidores HTTP. Dela fazem parte os seguintes atributos e métodos.

**Table 9.** Atributos da classe TCP

Atributo	Função
private Socket clientSocket	Variavel que contém o socket com o cliente.
private Socket serverSocket	Variavel que contém o socket com o servidor HTTP.
private static final int BUFFSIZE	Varivavel que define o tamanho do buffer usado como intermediário para o redireccionamento das streams entre sockets.
private final String endereco	Variavel que contém o endereço do servidor HTTP ao qual se vai estabelecer ligação.
private final int porta	Variavel que contém o numero da porta a usar para o protocolo TCP.
private InputStream inC	Stream de input do Cliente.
private OutputStream outC	Stream de output do Cliente.
private InputStream inS	Stream de input do Servidor.
private OutputStream outs	Stream de output do Servidor.
private boolean status	Variável responsável por manter ou terminar o processo de escuta do socket TCP.

**Table 10.** Métodos da classe TCP

Método	Função
public TCP(Socket socket, int p, String s)	Método que inicia uma nova instancia da classe TCP
public void run()	Método principal da classe TCP. É responsável por interligar um cliente a um servidor HTTP e manter a ligação ativa até que o cliente ou o servidor terminem a ligação.
private void pipeSocket(InputStream streamFrom, OutputStream streamTo)	Método que redirecciona o input de um socket para output de outro socket

**maquinas.** A classe máquinas é responsável por armazenar a informação relativa a cada servidor HTTP. Por questões de espaço os métodos get(), set(), hash() e equals() não foram incluídos neste documento, além destes parte os seguintes atributos e métodos.

**Table 11.** Atributos da classe maquinas

Atributo	Função
private String address	Endereço do servidor HTTP.
private int port	Porta do servidor HTTP.
private int counter	Variavel que guarda o numero total de pedidos de informação enviados

private int nReceived	Variavel que guarda o número total de respostas aos pedido de informação enviados.
private int nLigacoes	Variavel que guarda o numero de ligação TCP activas no servidor HTTP.
private long lastPong	Variavel que guarda o timestamp do ultimo PDU de status enviado pelo servidor HTTP.
private long[] receiveTime	Array que guarda o Round Trip Time dos últimos 20 pedidos de informação recebidos
private int[] smallLost	Array que guarda o estado de entrega ou não entrega dos últimos 20 pedidos de informação efetuados.
private int pos	Variavel que guarda a posição do índice a escrever no array receiveTime.

**Table 12.** Métodos da classe maquinas

Métodos	Função
maquinas(String adr, int prt)	Método que inicia uma nova instancia da classe maquinas

## 4 Testes e resultados

No decorrer do projeto, foram efetuados os seguintes testes ao servidor

### 4.1 Teste de carga ao proxy

Servidor 10.0.14.20, Ping#: 116, Perdas: 7.4766355%, RTT: 33.0ms, Ligacoes: 0.0	com pontuacao: 38.25037, Perdas: 60.34713%, RTT: 10.410095%, Ligacoes: 0.0%
Servidor 10.0.13.21, Ping#: 135, Perdas: 7.2%, RTT: 40.0ms, Ligacoes: 0.0	com pontuacao: 37.33237, Perdas: 59.114265%, RTT: 12.61826%, Ligacoes: 0.0%
Servidor 10.0.14.21, Ping#: 121, Perdas: 7.1428576%, RTT: 42.0ms, Ligacoes: 0.0	com pontuacao: 37.241682, Perdas: 57.653065%, RTT: 13.248211%, Ligacoes: 0.0%
Servidor 10.0.0.11, Ping#: 151, Perdas: 1.5513514%, RTT: 38.0ms, Ligacoes: 0.0	com pontuacao: 8.941878, Perdas: 10.307335%, RTT: 11.987381%, Ligacoes: 0.0%
Servidor 10.0.5.10, Ping#: 142, Perdas: 0.0%, RTT: 317.0ms, Ligacoes: 0.0	com pontuacao: 20.0%, Perdas: 0.0%, RTT: 100.0%, Ligacoes: 0.0%
Servidor 10.0.0.10, Ping#: 155, Perdas: 10.0%, RTT: 23.0ms, Ligacoes: 0.0	com pontuacao: 49.879677%, Perdas: 69.71428%, RTT: 7.255521%, Ligacoes: 0.0%
Maquina 10.0.0.11 com menos load,	
Servidor 10.0.13.20, Ping#: 148, Perdas: 11.363837%, RTT: 36.0ms, Ligacoes: 0.0	com pontuacao: 62.271297%, Perdas: 100.0%, RTT: 11.356467%, Ligacoes: 0.0%
Servidor 10.0.14.20, Ping#: 135, Perdas: 9.0%, RTT: 33.0ms, Ligacoes: 0.0	com pontuacao: 44.322018%, Perdas: 70.39894%, RTT: 10.410095%, Ligacoes: 0.0%
Servidor 10.0.13.21, Ping#: 155, Perdas: 6.2068963%, RTT: 40.0ms, Ligacoes: 0.0	com pontuacao: 35.236072, Perdas: 54.620682%, RTT: 12.61826%, Ligacoes: 0.0%
Servidor 10.0.14.21, Ping#: 141, Perdas: 6.060606%, RTT: 42.0ms, Ligacoes: 0.0	com pontuacao: 34.64984%, Perdas: 53.33333%, RTT: 13.248211%, Ligacoes: 0.0%
Servidor 10.0.0.11, Ping#: 171, Perdas: 1.764072%, RTT: 38.0ms, Ligacoes: 0.0	com pontuacao: 11.622506%, Perdas: 15.068835%, RTT: 11.987381%, Ligacoes: 0.0%
Servidor 10.0.5.10, Ping#: 162, Perdas: 0.0%, RTT: 317.0ms, Ligacoes: 0.0	com pontuacao: 20.0%, Perdas: 0.0%, RTT: 100.0%, Ligacoes: 0.0%
Servidor 10.0.0.10, Ping#: 175, Perdas: 10.126582%, RTT: 23.0ms, Ligacoes: 0.0	com pontuacao: 54.31946%, Perdas: 69.11332%, RTT: 7.255521%, Ligacoes: 0.0%
Maquina 10.0.0.11 com menos load,	
Servidor 10.0.13.20, Ping#: 158, Perdas: 13.605443%, RTT: 36.0ms, Ligacoes: 0.0	com pontuacao: 62.271297%, Perdas: 100.0%, RTT: 11.356467%, Ligacoes: 0.0%
Servidor 10.0.14.20, Ping#: 156, Perdas: 6.8865516%, RTT: 33.0ms, Ligacoes: 0.0	com pontuacao: 32.45681%, Perdas: 50.68865%, RTT: 10.410095%, Ligacoes: 0.0%
Servidor 10.0.13.21, Ping#: 175, Perdas: 5.4545453%, RTT: 39.0ms, Ligacoes: 0.0	com pontuacao: 26.515114%, Perdas: 40.089082%, RTT: 12.30283%, Ligacoes: 0.0%
Servidor 10.0.14.21, Ping#: 151, Perdas: 7.3825522%, RTT: 42.0ms, Ligacoes: 0.0	com pontuacao: 35.20626%, Perdas: 54.651739%, RTT: 13.248211%, Ligacoes: 0.0%
Servidor 10.0.0.11, Ping#: 191, Perdas: 1.604278%, RTT: 37.0ms, Ligacoes: 0.0	com pontuacao: 9.40925%, Perdas: 11.791442%, RTT: 11.671925%, Ligacoes: 0.0%
Servidor 10.0.5.10, Ping#: 182, Perdas: 0.0%, RTT: 317.0ms, Ligacoes: 0.0	com pontuacao: 20.0%, Perdas: 0.0%, RTT: 100.0%, Ligacoes: 0.0%
Servidor 10.0.0.10, Ping#: 155, Perdas: 9.60452%, RTT: 23.0ms, Ligacoes: 0.0	com pontuacao: 45.807034%, Perdas: 70.533216%, RTT: 7.255521%, Ligacoes: 0.0%
Maquina 10.0.0.11 com menos load,	

**Fig. 4.** Proxy a gerir 7 servidores HTTP, calculando as suas pontuações

## 4.2 Teste de clientes HTTP

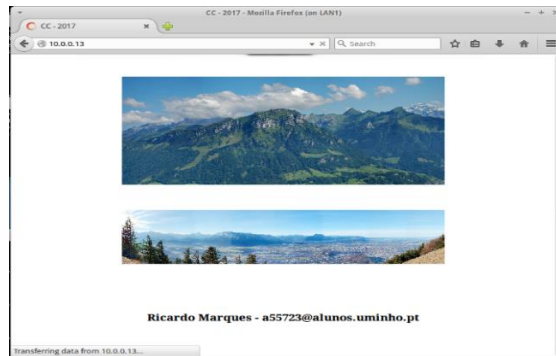


Fig. 5. Teste com o firefox

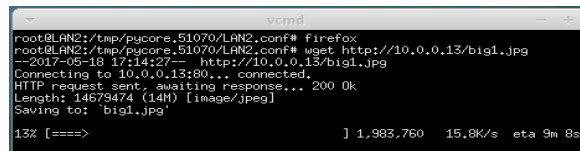


Fig. 6. Teste com o wget

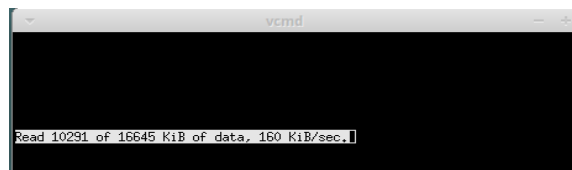


Fig. 7. Teste com o lynx

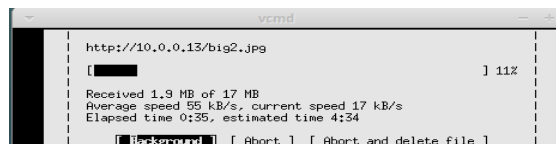
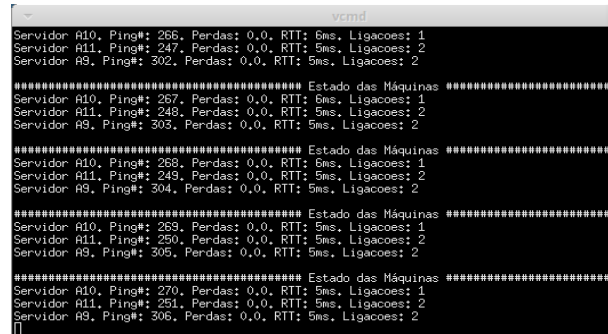


Fig. 8. Teste com o links2

### 4.3 Teste de funcionamento com múltiplos clientes e múltiplos servidores



```

vcmid
Servidor A10. Ping#: 266. Perdas: 0,0. RTT: 5ms. Ligacoes: 1
Servidor A11. Ping#: 247. Perdas: 0,0. RTT: 5ms. Ligacoes: 2
Servidor A9. Ping#: 302. Perdas: 0,0. RTT: 5ms. Ligacoes: 2

##### Estado das Máquinas #####
Servidor A10. Ping#: 267. Perdas: 0,0. RTT: 5ms. Ligacoes: 1
Servidor A11. Ping#: 248. Perdas: 0,0. RTT: 5ms. Ligacoes: 2
Servidor A9. Ping#: 303. Perdas: 0,0. RTT: 5ms. Ligacoes: 2

##### Estado das Máquinas #####
Servidor A10. Ping#: 268. Perdas: 0,0. RTT: 5ms. Ligacoes: 1
Servidor A11. Ping#: 249. Perdas: 0,0. RTT: 5ms. Ligacoes: 2
Servidor A9. Ping#: 304. Perdas: 0,0. RTT: 5ms. Ligacoes: 2

##### Estado das Máquinas #####
Servidor A10. Ping#: 269. Perdas: 0,0. RTT: 5ms. Ligacoes: 1
Servidor A11. Ping#: 250. Perdas: 0,0. RTT: 5ms. Ligacoes: 2
Servidor A9. Ping#: 305. Perdas: 0,0. RTT: 5ms. Ligacoes: 2

##### Estado das Máquinas #####
Servidor A10. Ping#: 270. Perdas: 0,0. RTT: 5ms. Ligacoes: 1
Servidor A11. Ping#: 251. Perdas: 0,0. RTT: 5ms. Ligacoes: 2
Servidor A9. Ping#: 306. Perdas: 0,0. RTT: 5ms. Ligacoes: 2

```

Fig. 9. Teste com 3 Servidores a responderem a vários pedidos em simultaneo.

## 5 Conclusões e trabalho futuro

Durante o desenvolvimento deste projeto, foi possível perceber o funcionamento dos protocolos UDP e TCP. Foi possível também compreender as enormes diferenças entre a utilização de ambos, quer a nível de implementação, quer a nível de interação entre aplicações.

Foi também possível compreender com mais profundidade a implementação de um reverse proxy transparente e notar que é uma ferramenta poderosa na gestão de uma rede.

No campo das redes propriamente ditas, foi possível apurar a dificuldade notória de controlar o estado de um link e de implementar medidas de atuem rapidamente para atenuar uma mudança repentina de estado, garantindo o máximo de performance e o mínimo tempo de resposta do proxy.

Futuramente planeia-se implementar um sistema de registo de atividade do proxy que resulte na criação de logs em texto.

Planeia-se também implementar um sistema de cache, para que pedidos efetuados recentemente sejam logo servidos a partir do proxy, e evitar pedidos repetidos aos servidores, diminuindo assim parte da carga na rede.