

# Balanceamento de carga com servidor proxy invertido

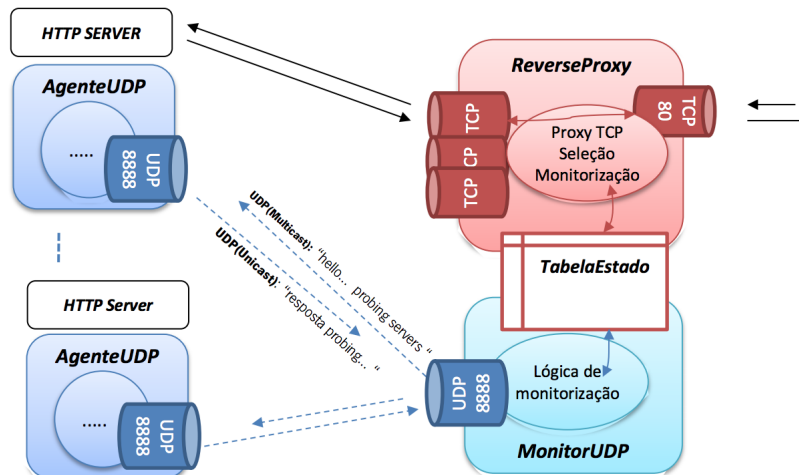
Rui Vieira, Filipe Fortunato, and Frederico Pinto

Universidade do Minho

**Resumo** Este relatório é feito no âmbito da disciplina de âmbito da disciplina de *Comunicação por Computador* relativo ao TP2 *Balanceamento de carga com servidor proxy invertido*. O principal objetivo deste trabalho é o de criar um servidor TCP invertido em que o servidor é escolhido através de alguns parâmetros como o **RTT**, **RAM** disponível e a percentagem de **CPU** não utilizada. Em primeiro lugar implementamos um protocolo **UDP** e criamos uma tabela que vai ser utilizada uma tabela com a informação do servidor. De seguida implementamos um servidor proxy **TCP**, que fica à escuta na porta **80** e re-direciona automaticamente as ligações que recebe para a mesma porta de um dos servidores disponíveis(o que estiver em melhores condições).

## 1 Arquitetura da solução

A arquitetura presente, é baseada na arquitetura inicialmente proposta. Após a análise do problema, foi adicionada a classe StatusTable, que contém nela as informações de um dado agente. É de notar que para escolher o melhor servidor o Reverse Proxy acessa a StatusTable retirando o melhor disponível, que se encontra na cabeça da lista.



## 2 Especificação do protocolo UDP

### 2.1 PDU - Formato das mensagens protocolares

Como sabemos, de maneira a ser possível armazenar a diferente informação para podermos calcular qual o servidor mais adequado tem que existir comunicação entre o monitor e os diferentes agentes. O transporte de pacotes entre os diferentes sistemas é utilizado o protocolo UDP. No nosso sistema, estes pacotes assumem dois tipos diferentes, **probe request** e **probe response** que se encontram em *JSON*.

#### – Monitor - Agente

Com base no que foi dito anteriormente, o formato geral de um pacote enviado do monitor para os agentes é:

- tipo - probe request

#### – Agente - Monitor

Com base no que foi dito anteriormente, o formato geral de um pacote enviado do agente para o monitor é:

- tipo - probe response
- RAM - total de memória RAM livre
- CPU - percentagem de CPU não utilizada
- IP - endereço IP do servidor
- Door - porta do servidor

Este pacote tem um formato mais extenso que o anterior, pois é necessário que o agente responda com a informação necessária de maneira a que o monitor consiga gerir a StatusTable colocando na primeira posição da mesma o melhor servidor disponível.

### 2.2 Interações

A comunicação é feita em dois tipos, *unicast* e *multicast*. O primeiro faz a comunicação apenas entre dois servidores, e o segundo faz a comunicação de um servidor com vários servidores. Assim, a comunicação entre o monitor e agentes é feita em multicast, para permitir fazer vários pedidos ao mesmo tempo e a inversa é feita em unicast, porque o agente apenas precisa de responder a um monitor.

## 3 Implementação

Após bastante reflexão optamos por realizar o trabalho em python, devido não só a simplicidade de syntax da mesma mas também devido às enumeras bibliotecas que simplificam a resolução do enunciado proposto.

### 3.1 Reverse Proxy

Nesta componente, é inicializada uma instância de um UDP Monitor que fica a correr à parte (Thread). De seguida, é criada o socket correspondente a entrada dos pedidos http, sendo que este encontra-se no endereço localhost, porta 80. Após esta ligação é iniciada uma espera por pedidos de clientes, onde posteriormente o ReverseProxy retira o melhor servidor disponível do StatusServer de seguida é criada uma Thread que liga os pedidos do cliente ao servidor selecionado. Essa mesma Thread cria outra de maneira a ser possível fazer o caminho inverso, isto é, haver comunicação do servidor para o cliente.

### 3.2 StatusTable

Esta componente contém um dicionário que usa como key um float, que é calculado a partir de da seguinte fórmula  $(ram * (1 - (cpu * 0.01)) * elapsedtime) / 3$  onde a ram e cpu representam o total de RAM livre e percentagem de CPU não utilizada respetivamente guardando como value o probe response dando append do RTT. Definindo métodos para atualizar o dicionário e retorno do melhor disponível.

### 3.3 UDPServer

A implementação desta componente do sistema encontra-se definida numa classe denominada **server**. Começamos por fazer o pedido para a entrada no grupo *multicast* de maneira a que um monitor possa comunicar com o agente. Após entrada neste grupo, o agente fica a espera para receber uma mensagem. Após receber a mensagem este processa a informação e caso a mensagem seja do tipo **probe request** o servidor guarda a informação da mensagem com uma biblioteca chamada **psutil**. Ao enviar os pacotes teve que ser considerado um intervalo de tempo aleatório de maneira a não enviar todos os pacotes de uma vez. O formato das mensagens é **JSON** sendo convertido para bytes aquando do envio.

### 3.4 Monitor

Monitor é uma classe, que ao ser inicializada cria o socket e guarda as informações da configuração e cria também duas *threads*. O Monitor usa estas *threads* para tarefas diferentes. A segunda é usada para o Monitor enviar a mensagem do tipo **probe request** a cada 5 segundos para o grupo *multicast* guardando o tempo a que esta foi enviada e a primeira serve para o Monitor estar à espera das mensagens do tipo **probe response** e aquando da receção deste tipo de mensagem este guarda o tempo a que recebeu a mensagem para depois calcular o tempo passado entre o envio da mensagem **probe request** e a receção da mensagem **probe response** e depois atualiza toda a informação sobre o Agente na **StatusTable**.

4

## 4 Testes e resultados

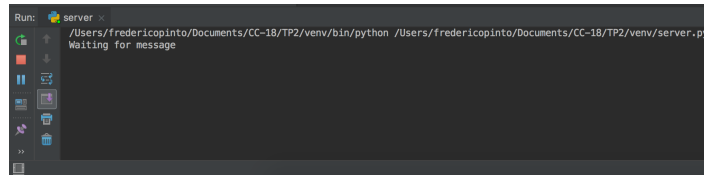


Figura 1. Server a espera de mensagens

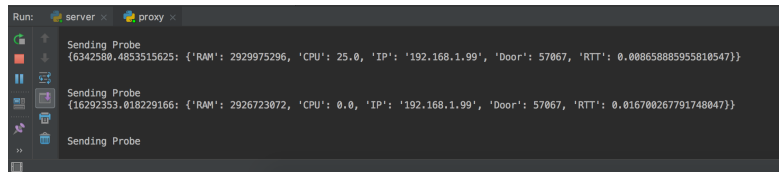


Figura 2. Proxy a funcionar com monitor

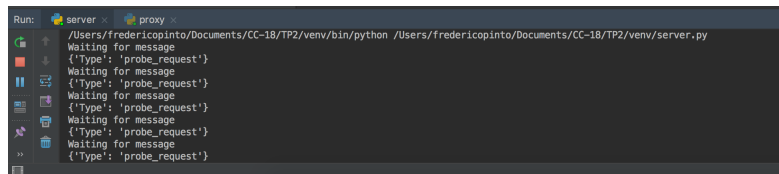


Figura 3. Server a imprimir a mensagem que recebe

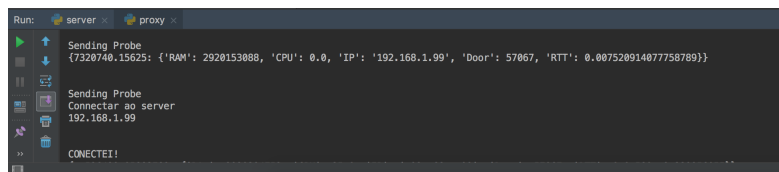
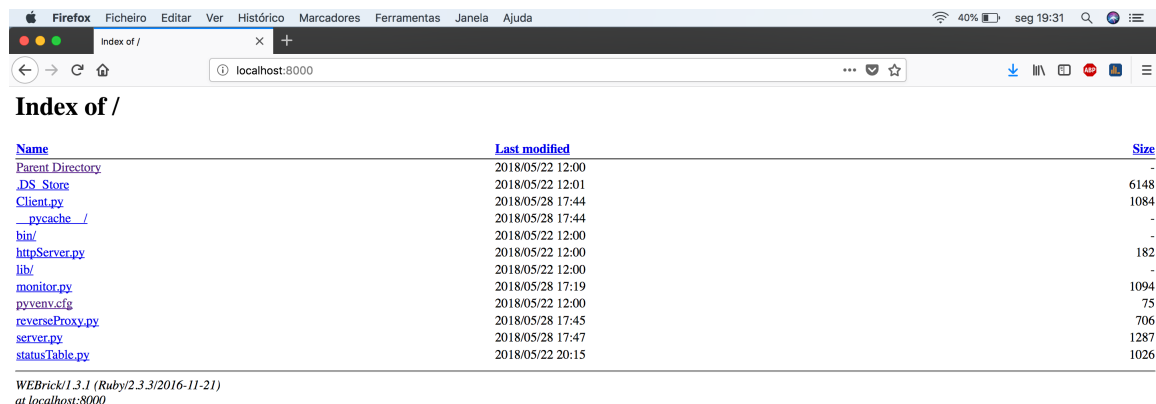


Figura 4. Cliente a conectar ao server



Name	Last modified	Size
<a href="#">Parent Directory</a>	2018/05/22 12:00	-
<a href="#">.DS_Store</a>	2018/05/22 12:01	6148
<a href="#">Client.py</a>	2018/05/28 17:44	1084
<a href="#">__pycache__/</a>	2018/05/28 17:44	-
<a href="#">bin/</a>	2018/05/22 12:00	-
<a href="#">httpServer.py</a>	2018/05/22 12:00	182
<a href="#">lib/</a>	2018/05/22 12:00	-
<a href="#">monitor.py</a>	2018/05/28 17:19	1094
<a href="#">pyvenv.cfg</a>	2018/05/22 12:00	75
<a href="#">reverseProxy.py</a>	2018/05/28 17:45	706
<a href="#">server.py</a>	2018/05/28 17:47	1287
<a href="#">statusTable.py</a>	2018/05/22 20:15	1026

WEBrick/1.3.1 (Ruby/2.3.3/2016-11-21)  
at localhost:8000

**Figura 5.** Resultado final

## 5 Conclusões e trabalho futuro

Este trabalho prático foi um pouco diferente daqueles que estávamos habituados a desenvolver na Unidade Curricular, no entanto, não foi por isso que deixou de ser interessante e apelativo ao grupo. Com a sua realização ficamos a saber bastante mais sobre o que são e como funcionam, de uma forma bastante técnica, os servidores de front e back-end.

Durante a sua elaboração deparamo-nos com várias incertezas e dificuldades que, depois de alguma dedicação e esforço, conseguimos ultrapassar, levando-nos a tomar várias decisões importantes para o rumo do projeto.

Assim, a elaboração deste trabalho prático revelou ser de extrema importância e bastante útil para todo o grupo, uma vez que nos permitiu pôr em prática os conhecimentos adquiridos durante as aulas e, ao mesmo tempo, aperfeiçoar e aprofundar outros conhecimentos e ferramentas das quais tiramos proveito, como por exemplo, a linguagem de programação python, enriquecendo-nos com uma experiência que certamente será bastante útil no futuro.

Concluindo, pensamos que o resultado final decorrente da resolução do trabalho proposto vai de encontro aquilo que era esperado.

## Bibliografia

- [1] Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. 78, 315–333 (1982)
- [2] Clarke, F., Ekeland, I.: Solutions périodiques, du période donnée, des équations hamiltoniennes. Note CRAS Paris 287, 1013–1015 (1978)
- [3] Michalek, R., Tarantello, G.: Subharmonic solutions with prescribed minimal period for nonautonomous Hamiltonian systems. J. Diff. Eq. 72, 28–55 (1988)
- [4] Tarantello, G.: Subharmonic solutions for Hamiltonian systems via a  $\mathbb{Z}_p$  pseudoindex theory. Annali di Matematica Pura (to appear)
- [5] Rabinowitz, P.: On subharmonic solutions of a Hamiltonian system. Comm. Pure Appl. Math. 33, 609–633 (1980)