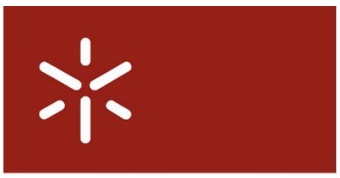


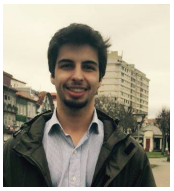
Universidade do Minho

Comunicações por Computador

MIEI - 3º ano - 2º semestre
Universidade do Minho

Trabalho prático

Proxy TCP reverso com monitorização proactiva



Dinis Peixoto
A75353

Ricardo Pereira
A74185

Marcelo Lima
A75210

16 de Novembro de 2017

Conteúdo

1. *Introdução*

Este relatório é relativo à elaboração do Trabalho Prático 2 (TP2) proposto na Unidade Curricular *Comunicações por Computador 2016/2017* com o tema *Proxy TCP reverso com monitorização proactiva*. Este trabalho foi elaborado ao longo de 5 aulas, dividido em duas fases, em que a primeira, com a duração de 3 aulas tem um peso de 14 valores e a segunda, com a duração de 2 aulas, tem um peso de 6 valores.

O principal objetivo deste trabalho é desenhar e implementar um serviço simples de proxy reverso TCP em que a escolha do servidor a usar se faz com base em parâmetros dinâmicos, como por exemplo o RTT, as perdas e número de conexões TCP do servidor.

Será necessário na primeira fase desenhar e implementar um protocolo sobre UDP para criar e manter atualizada uma tabela com dados/medidas recolhidas por servidor.

Depois, numa segunda fase, pretende-se implementar um servidor proxy TCP genérico, que fique à escuta na porta 80 e redirecione automaticamente cada conexão TCP na porta 80 que receber para a mesma porta de um dos servidores disponíveis (o que aparente estar em melhores condições para o fazer).

Desta forma, ao longo deste relatório iremos expor e explicar todo o processo que tivemos de percorrer até chegarmos ao resultado final, assim como todas as decisões que tivemos de tomar durante a sua execução.

2. *Arquitetura da solução implementada*

Para o desenvolvimento do problema apresentado foi necessário logo à partida estabelecer a arquitetura que o nosso *ReverseProxy* e Monitor iriam apresentar, de forma a permitir usar todas as funcionalidades requisitadas. Desde modo, segue-se a arquitetura final implementada:

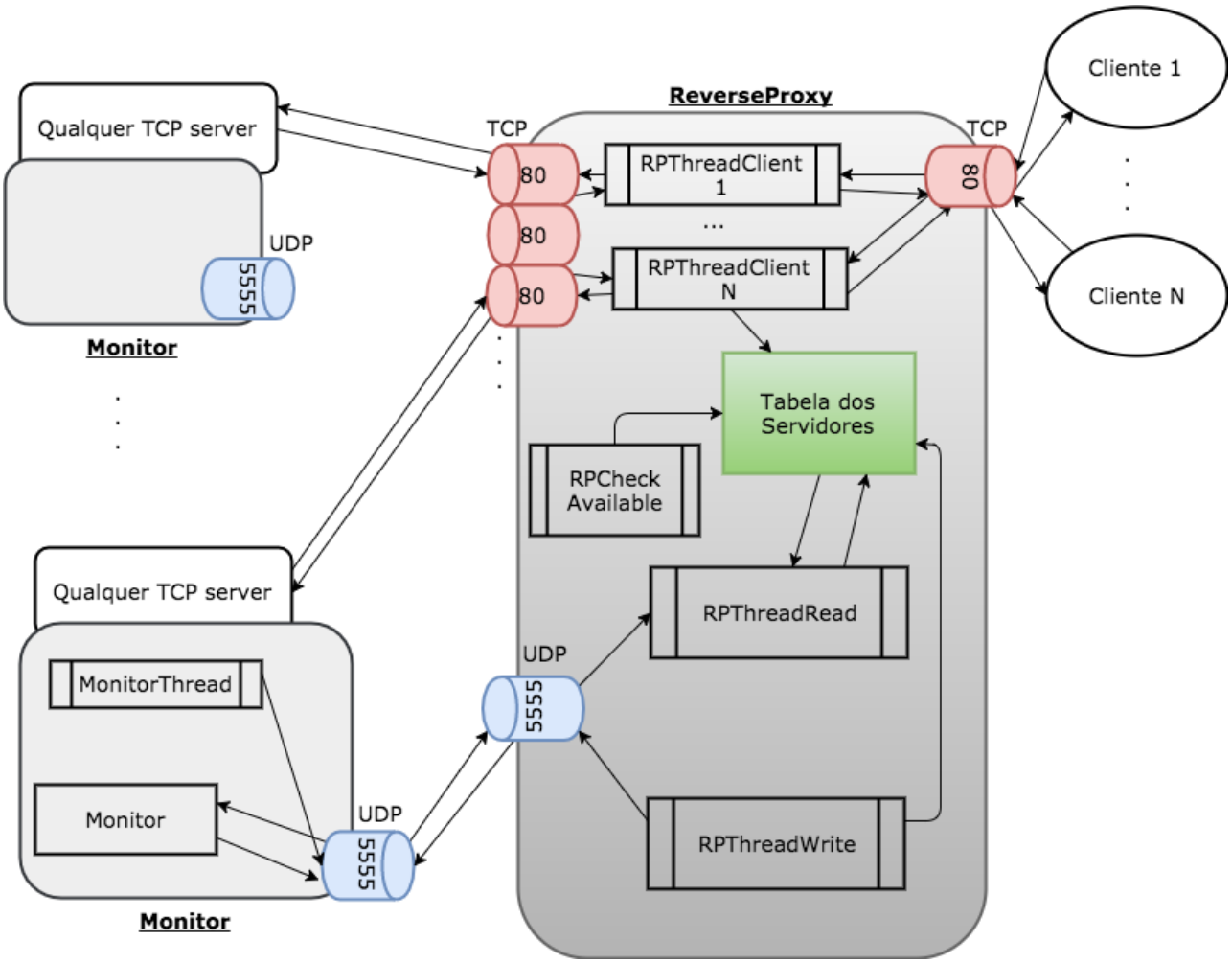


Figura 2.1: Arquitetura Geral.

2.1 *ReverseProxy*

Do lado do servidor *ReverseProxy*, para atender às necessidades dos processos de monitorização, criamos 3 *Threads* de forma a que o processamento seja simultâneo.

- **RPThreadRead** - Esta *thread* é responsável por ler todos os pacotes recebidos por via UDP (porta 5555), e actualizar a respectiva tabela com as informações dos utilizadores.
- **RPThreadWrite** - Como é de esperar, esta *thread* é responsável unicamente por enviar os pacotes de *"Probing"*, para os vários servidores de *back-end*, ligados no momento. Este envio é realizado de 3 em 3 segundos, sendo que, ao fim de cada intervalo, envia os pacotes a todos os servidores, isto continuamente e repetitivamente.
- **RPCheckAvailable** - De forma a garantir o bom funcionamento de todo o sistema, é necessário garantir que, um dado servidor se encontra disponível ou não. Para tal, esta *thread*, verifica de 60 em 60 segundos, todos os servidores, procurando saber se estes enviaram um pacote de disponibilidade dentro do tempo limite considerado.

2.1.1 Tabela dos Servidores

Para garantir uma monotorização correta e adqueada foi necessária a criação de uma tabela, cujo objetivo passaria por guardar informações estatísticas dos servidores de *back-end*, de forma a que o servidor *ReverseProxy* possa saber qual o melhor servidor num dado momento. Como tal, para cada servidor a tabela possui os seguintes parâmetros:

- **Endereço IP** - endereço IP do servidor *back-end*;
- **Porta** - porta a ser usada na conexão (neste problema é sempre a 5555);
- **RTT** - *Round Trip Time* (valor decimal);
- **Taxa de Pacotes Perdidos** - percentagem de pacotes perdidos (0-100%);
- **Nº do pacote** - número do próximo pacote a ser enviado;
- **Nº do pacote verificado** - número do pacote já recebido;
- **Nº de pacotes perdidos** - número de pacotes perdidos;
- **Última verificação** - valor inteiro (*long*), a que corresponde um *timestamp*;
- **Média** - média dos 3 parâmetros a ser avaliados;
- **Disponível** - valor booleano, indicando a disponibilidade do Servidor;
- **Nº conexões TCP** - número de conexões TCP activas no momento ao servidor.

Apresentaremos de seguida o exemplo de uma tabela com valores válidos para uma situação arbitrária.

Parâmetros	Servidor 1 (...)	Servidor N
Endereço IP	10.0.1.10	10.0.3.11
Porta	5555	5555
RTT	2.54	1.43
Taxa de Pac. Perdidos	13.43%	18.32%
Nº do Pacote	43	23
Nº do Pacote verificado	42	23
Nº de Pacotes perdidos	6	4
Última verificação	1494867099409	1494867099417
Média (Classificação)	5.99	6.92
Disponível	True	True
Nº conexões TCP	2	1

Figura 2.2: Tabela dos Servidor de *back-end*.

2.2 Monitor

Por sua vez, do lado do servidor de *back-end*, o respectivo monitor para comunicar e trocar informações com o servidor *ReverseProxy* também necessita da criação de *Threads*, no entanto, apenas é criada uma *Thread* nova. O mecanismo do Monitor baseia-se na *Thread* e no respectivo processo Monitor que acarretam a responsabilidade de todo o funcionamento da monitorização do lado do servidor *back-end*. Sendo assim, temos:

- **Monitor** - responsável por receber um pacote via UDP (porta 5555), e re-enviar para a orgiem deste. Desta forma, podemos medir alguns parâmetros de monitorização, que abordaremos mais à frente.
- **MonitorThread** - responsável por enviar pacotes do tipo *"Available"* via UDP (porta 5555) de 5 em 5 segundos. Como é de esperar, caso esta *thread* pare de correr, todo o servidor *back-end* associado torna-se indisponível para o servidor *ReverseProxy*.

3. *Especificação do protocolo de monitorização*

3.1 Pacotes

Os pacotes enviados por via UDP, tanto do tipo *"Available"* como do tipo *"Probing"* possuem uma estrutura muito idêntica sendo distinguidos apenas por uns meros parâmetros, como tal regra geral um pacote é constituído da seguinte forma:

- **Endereço IP** - endereço IP do servidor que envia o pacote;
- **Porta** - porta a ser usada na conexão (neste probelma é sempre a 5555);
- **Tipo** - indica se é do tipo *"Available"* ou *"Probing"*;
- **Tempo de saída** - valor inteiro que corresponde ao timestamp do momento de saida do pacote do servidor;
- **Nº do pacote** - número de série do pacote. Caso o pacote seja do tipo *"Available"*, o número é sempre -1.

Os pacotes do tipo *"Available"* apenas possuem um sentido, isto é, do servidor de *back-end* até ao servidor *ReverseProxy*. No entanto, os pacotes do tipo *"Probing"*, possuem os dois sentidos, saindo sempre do servidor *ReverseProxy*, passando posteriormente pelo Monitor de um servidor de *back-end*, e voltando por fim para o servidor de origem.

3.2 RTT

O *Round Trip Time* (RTT) corresponde ao tempo que um pacote demora a ir ao monitor de um servidor *back-end* e voltar ao servidor *ReverseProxy*. Para o cálculo do RTT, foi utilizada a seguinte expressão:

$$RTT = ((1 - alfa) * RTT + alfa * (TempoActual - TempoSaida))$$

Com,
alfa = 0.125;
TempoActual - *timestamp* no preciso momento;
TempoSaida - *timestamp* no momento de saida do pacote da *RPThreadWrite*.

3.2.1 Decisões

O valor do RTT é atualizado unicamente com os pacotes do tipo *"Probing"* recebidos de volta. Os pacotes que eventualmente se perdem nada afetam o valor deste, uma vez que este é atualizado com o valor de saída do pacote, com o tempo no momento.

3.3 Taxa de pacotes Perdidos

Para calcular a taxa de pacotes perdidos recorremos aos pacotes do tipo *"Probing"* uma vez que são estes que nos fornecem os dados necessários. O pacote do tipo *"Probing"* no momento do envio (por parte da *RPThreadWrite* do *ReverereseProxy*), indexa o pacote com o número de série em que se encontra, no processo de *probing*, permitindo assim que a *RPThreadRead* no momento da leitura do pacote de retorno deste possa efitivamente verificar se o número de série do pacote é igual ao esperado. Caso o número de série recebido seja maior do que o esperado a *RPThreadRead* actuliza a taxa de pacotes perdidos da segundo o seguinte método:

$$NumPacoteCheck = NumPacote + 1$$

$$taxPacLost = \frac{falhas + diferença}{NumPacote} * 100$$

- **NumPacoteCheck** - número do pacote à espera de ser recebido.
- **NumPacote** - número do pacote recebido.
- **taxPacLost** - taxa de pacote perdidos em percentagem.
- **falhas** - número de pacotes já perdidos.
- **diferença** - diferença entre *NumPacoteCheck* e *NumPacote*.

Para o caso de não ocorrer nenhuma falha, o RPThreadRead actualiza o NumPacoteCheck, incrementando-o apenas.

3.3.1 Decisões

Relativamente a este ponto, uma das decisões tomadas, foi considerar o intervalo de tempo de saída e o número de pacotes a enviar de cada vez como forma de balancear a subcarga da rede. Para tal, o intervalo de tempo estabelecido é de 3 em 3 segundos dos pacotes do tipo *"Probing"*, e limitamos este a 1 pacote enviado de cada vez, garantindo que o método acima apresentado permite obter os pacotes perdidos com um envio exclusivo de 1 pacote. Assim, conseguimos realizar uma monotorização relativamente leve para a rede em geral.

3.4 Número de conexões TCP

O cálculo do número de conexões TCP passa apenas por incrementar o número de conexões aquando uma nova ligação TCP e o respectivo decremento no fim da ligação. Desta forma conseguimos facilmente manter o número de conexões sempre atualizado.

3.4.1 Decisões

Neste parâmetro consideramos que o número de conexões TCP, correspondiam ao número de conexões activas no momento com o servidor, podendo assim medir a carga do respectivo servidor.

3.5 Média

A média consiste basicamente no cálculo da média dos 3 seguintes critérios:

- **RTT**
- **Taxa de pacotes perdidos**
- **Número de conexões TCP**

3.5.1 Decisões

Dado que os 3 critérios de avaliação possuiem todos um peso importante, dado que por isso consideramos em atribuir a todos os crtérios o mesmo peso, sendo por isso a média o nosso crtério de escolha de um servidor *back-end*.

3.6 Disponibilidade

Para verificar a disponibilidade dos servidores *back-end*, possuimos a RPCheckAvailable que a cada minuto verifica a tabela com as informações dos servidores *back-end*, verificando o parâmetro do último envio do pacote do tipo *"Available"*. A verificação consiste em ver se caso o *timestamp* do servidor é maior ou igual a 15 segundos (o que corresponde ao tempo de envio 3 pacotes do tipo *"Available"*), o RPCheckAvailable passa a disponibilidade do servidor *back-end* para falso.

3.6.1 Decisões

Dado que o RPCheckAvailable faz a verificação dos servidores em 1 em 1 minuto e um servidor *back-end* responde 5 em 5 segundos com um pacote do tipo *"Available"*, o limite de 15 segundos é um tempo limite suficientemente aceitável, dado que podem ocorrer evenutais pacotes perdidos.

4. *Demostração do funcionamento*

Neste capítulo iremos explicar, passo a passo, todo o processo de execução do proxy TCP reverso, de modo a podermos, desta forma, testar o seu funcionamento, com vários servidores back-end e clientes.

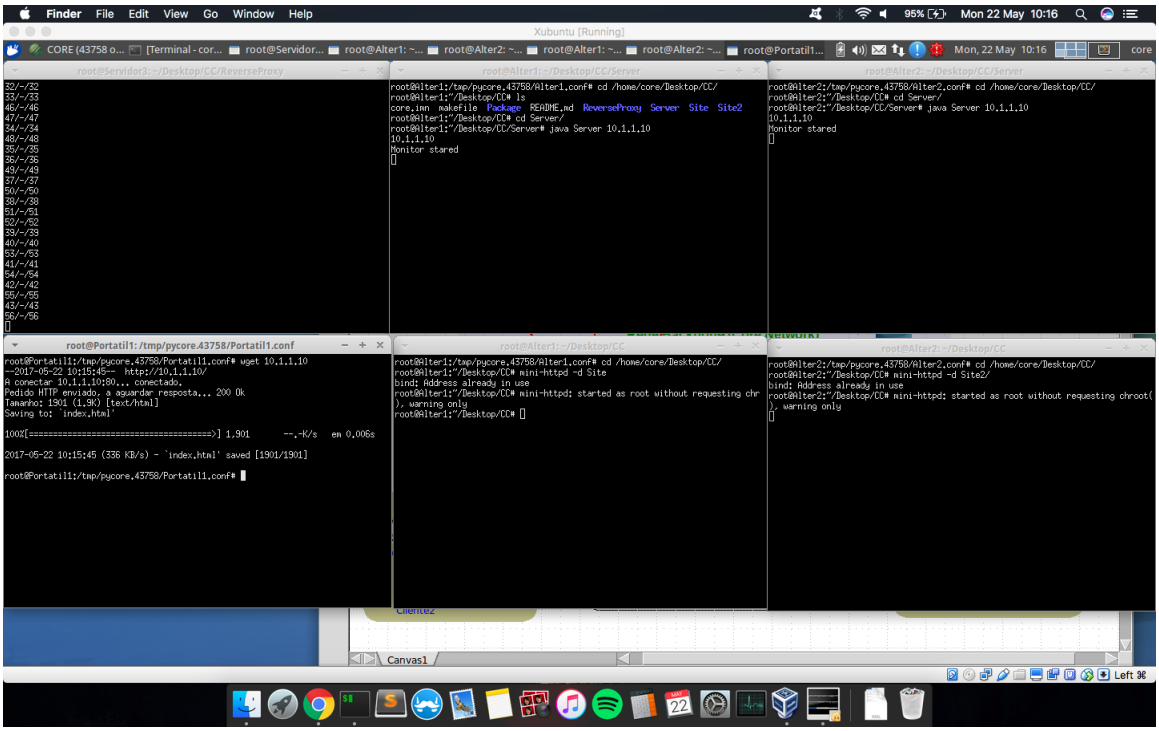


Figura 4.1: Execução geral de todo o sistema.

Esta é a vista que iremos obter depois de iniciarmos a execução do proxy TCP reverso juntamente com dois servidores back-end com os respetivos monitores, e ainda um cliente a tentar aceder ao mesmo.

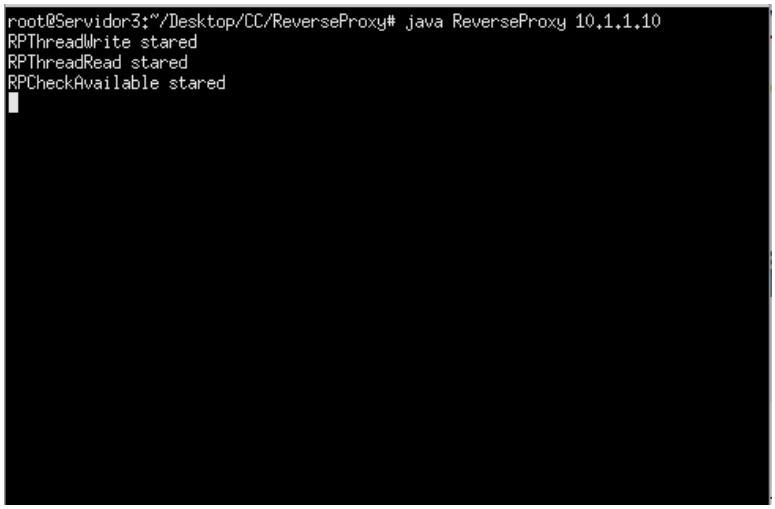


Figura 4.2: Execução do reverse proxy.

Para ligarmos o reverse proxy basta selecionar um dos servidores existentes no core e, de seguida, executar a respetiva aplicação seguido do ip que este assume na topologia. Como é possível observar na imagem, após este ser executado, são imediatamente iniciadas as três threads: RPThreadWrite, RPThreadRead e RPCheckAvailable.

Figura 4.3: Execução de um servidor back-end.

Para ligarmos um servidor back-end ao reverse proxy basta selecionar um servidor que se encontre na mesma topologia que este e, de seguida, executar a respetiva aplicação seguido do ip do reverse proxy anteriormente definido. Como é possível observar na imagem, após este ser executado, é imediatamente iniciada a thread Monitor.

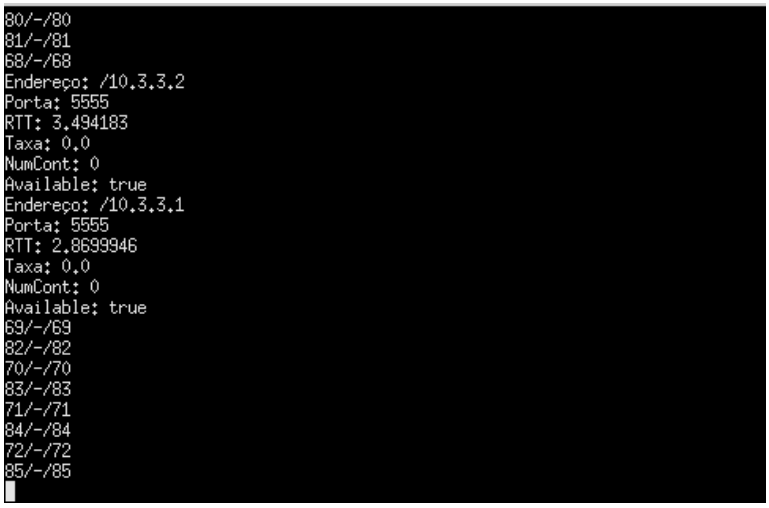


Figura 4.4: Informações fornecidas pelo monitor.

Logo após ocorrer a ligação de um servidor back-end ao reverse proxy é mostrada uma série de informações relativas aos pacotes de probing, assim como várias outras informações associadas a esse mesmo servidor, fornecidas pelo monitor.

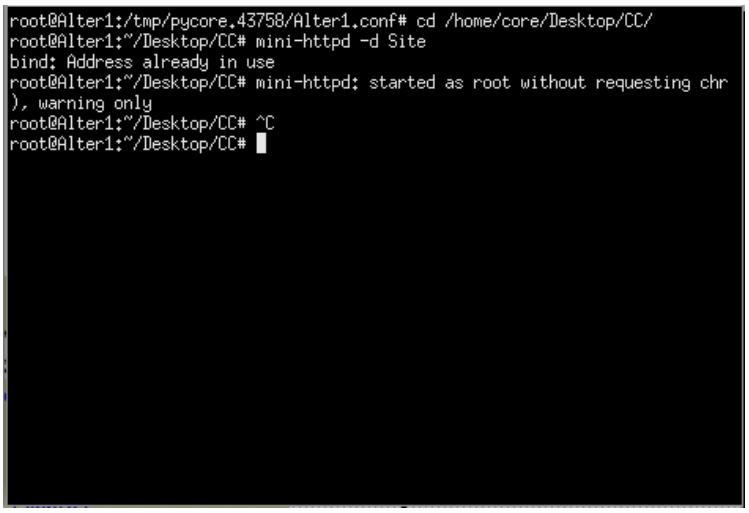


Figura 4.5: Execução do mini-http no servidor back-end.

De forma a podermos testar de maneira mais rigorosa todo o processo de ligação e troca entre os clientes e os servidores, executamos um mini-http no servidor back-end, que posteriormente tentará ser acedido por um cliente. Para isso basta executar o mini-http seguido da respetiva diretoria onde este se encontra armazenado.

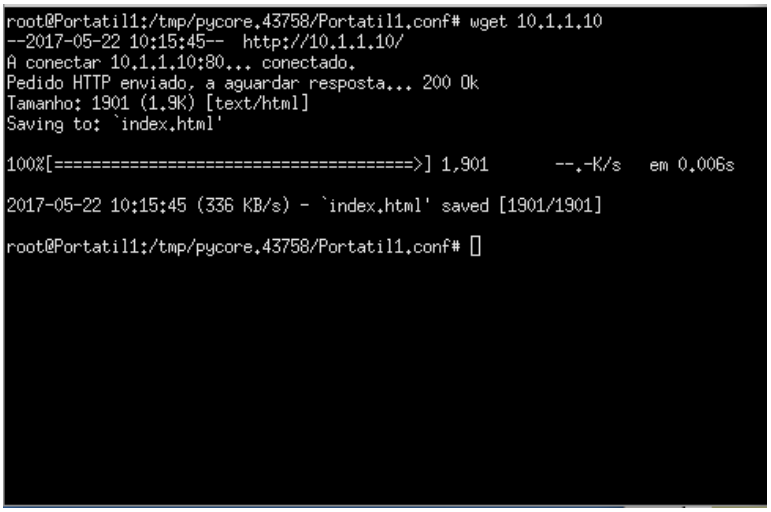


Figura 4.6: Pedido do cliente ao reverse proxy.

Agora sim já estamos aptos a ligar um cliente ao reverse proxy de forma a aceder ao mini-http presente nos servidores back-end. Para isso basta executar um **wget** seguido do ip do reverse proxy.

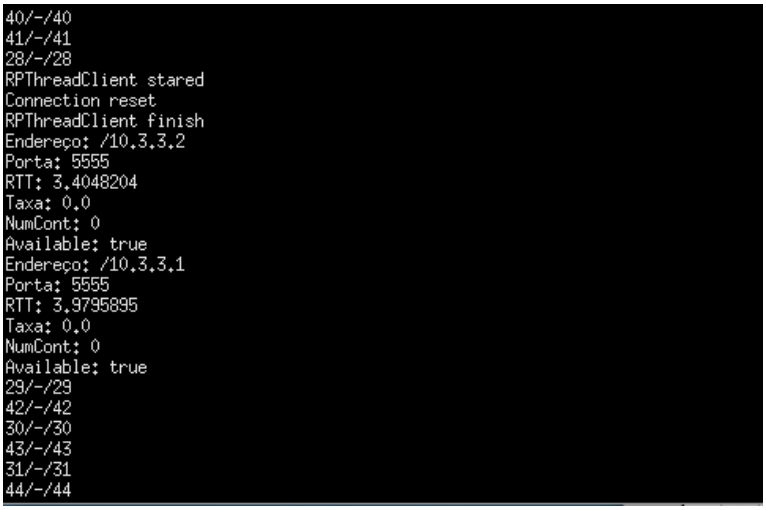


Figura 4.7: Ligação do cliente.

Após o cliente fazer o pedido é mostrado no terminal do reverse proxy a ligação deste ao mesmo com a respetiva RPThreadClient.

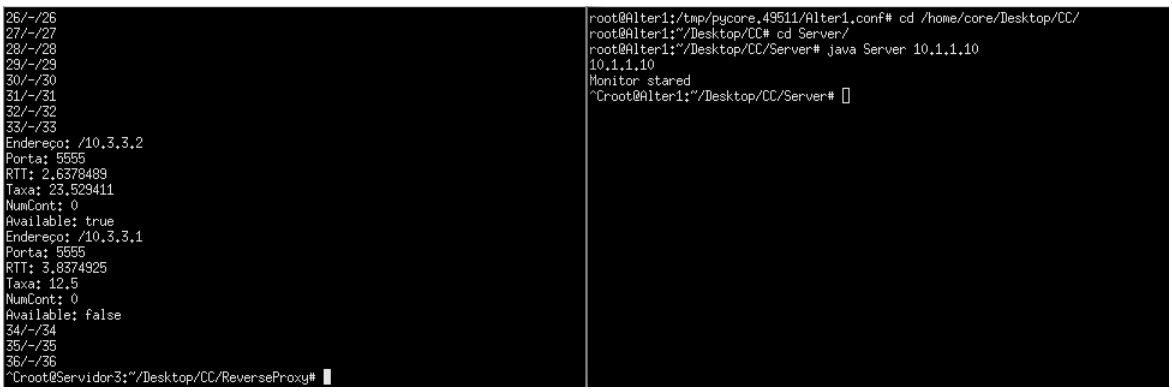


Figura 4.8: Monitor desligado.

A figura acima, mostra quando um monitor para de correr, o respectivo ReverseProxy (através da RPCheckAvailable que verifica a disponibilidade dos servidores *back-end*), passa o campo *Available* para false.

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 Java e o emulador core, 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, na medida em que nos empenhamos da melhor maneira possível.