Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Reverse Proxy - TP2

Comunicações por Computador

Grupo de trabalho (PL38)
André Santos, A61778
Diogo Machado, A75399
Rui Leite, A75551

Braga, 29 de Maio de 2017

Conteúdo

1. Introdução

Um *proxy* é uma entidade que tipicamente serve de intermediário numa comunicação entre um cliente e um servidor. Podemos distinguir dois tipos de proxys: *forward-proxy* e *reverse-proxy*. Num *forward-proxy*, existem vários clientes associados ao proxy, que através dele podem efetuar pedidos a potencialmente qualquer servidor. Num *reverse-proxy* são os servidores que estão associados ao proxy, cujos conteúdos que podem ser acedidos potencialmente por um qualquer cliente. Os *reverse-proxy* podem ser usados para esconder características dos servidores onde o conteúdo se encontra ou até mesmo esconder a existência dos servidores. Podem também ser usados para fazer *caching* de conteúdos e assim reduzir o tráfego aos servidores associados.

Neste trabalho pretende-se desenvolver um sistema capaz de simular o funcionamento de um *reverse-proxy*. A implementação do *reverse-proxy* propriamente dito será feita em Java. Os *WebServers* associados ao *reverse-proxy* comunicarão com *reverse-proxy* para efeitos de monitorização através de sockets implementados em Java, sendo os conteúdos servidos pelo servidor de páginas HTTP *mini-httpd*. A arquitetura geral do sistema será vista na secção 2 e na secção 3 será explicada de que forma o *reverse-proxy* monitoriza os servidores associados e de que forma esta monitorização é importante para que o *proxy* possa escolher de que servidor deve obter o conteúdo pedido pelos clientes. Na secção 4 serão explicados os detalhes de implementação envolvidos no desenvolvimento deste sistema. Por fim, na secção 5 serão apresentados alguns testes e resultados que demonstram as funcionalidades implementadas do sistema.

2. Arquitectura da solução

O sistema desenvolvido pode ser separado em dois componentes distintos: de um lado os agentes de monitorização dos servidores de *back-end* e de outro o servidor de *front-end*, capaz de intermediar pedidos de clientes.

A monitorização dos servidores de *back-end*, os *Web Servers*, opera sobre conexões UDP na porta 5555. Cada um dos *Web Servers* tem a si associado um monitor que tem como simples função a de enviar, periodicamente, um anúncio da disponibilidade do seu *Web Server* ao gestor de monitorização do *Reverse Proxy* (monitorização passiva). Além disso, terá que, também periodicamente, receber pedidos de *probing* do *Reverse Proxy* e responder-lhe com um *probe reply* (monitorização activa). Todas estas comunicações entre servidor *front-end* e servidor *back-end* são usadas na avaliação da disponibilidade e da qualidade de cada um dos *Web Servers*.

A intermediação dos pedidos dos clientes está assente em comunicações TCP. Sempre que chega um pedido de um cliente ao servidor, este terá que escolher o melhor *Web Server* disponível no momento e enviar o pedido do cliente ao respetivo. Depois, ao receber a resposta, reencaminha-a ao cliente.

A escolha do melhor *Web Server* para responder ao cliente é feita tendo por base todas as avaliações e registos feitos até ao momento, quer com os pedidos periódicos de monitorização já referidos, quer com a resposta aos pedidos dos clientes. Todos esses registos são guardados numa tabela de monitorização mantida pelo *Reverse Proxy* e contém informações como o *RTT*, o número de conexões e a taxa de pacotes perdidos. No capítulo 3 deste relatório é feita a apresentação detalhada dos registos mantidos.

Na Figura 2.1 poderá ser analisada a arquitetura geral do sistema, com as entidades principais envolvidas e as comunicações entre elas.

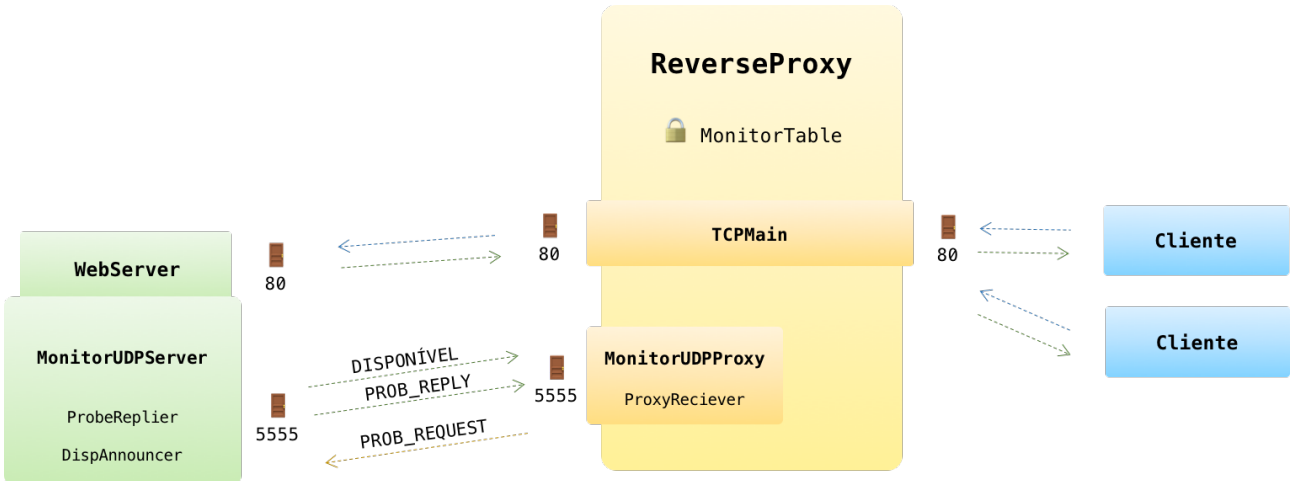

Figura 2.1: Arquitetura do sistema

3. Protocolo de Monitorização

De forma a poder monitorizar o estado dos servidores são trocadas mensagens entre o `MonitorUDP` e o `ReverseProxy`. Esta comunicação é feita recorrendo ao protocolo UDP e é realizado nas portas já apresentadas na imagem 2.1.

3.1 *Protocol Data Unit*

Por forma a efetuar a comunicação protocolar foi criada uma PDU com os seguintes campos: número de sequência, o tipo da PDU e o instante em que foi enviada a mensagem. O número de sequência corresponde a um inteiro incremental usado na monitorização ativa dos WebServers e como iremos ver de seguida servirá para a detecção de pacotes perdidos. O tipo da PDU identifica o tipo de mensagem da PDU. Trata-se de uma enumeração em Java e pode tomar um dos seguintes valores: DISPONIVEL, para monitorização passiva, ou PROB_REQUEST e PROB_REPLY para monitorizaçao activa. Por fim, o tempo em que a trama foi enviada é utilizado sobretudo pelo ReverseProxy para actualização do tempo em que um WebServer se mostrou disponível pela última vez.

3.2 Tipo de interações

Quando o `MonitorUDP` é iniciado são enviadas mensagens periódicas de 3 em 3 segundos para o `ReverseProxy`. Estas mensagens são do tipo `DISPONIVEL`. Neste tipo de mensagens o número de sequência não é relevante, pelo que deve ser ignorado ao ler este tipo de pedido. Além disso estas mensagens contêm o tempo em que foram enviadas. Este tempo de envio é importante para informar ao `ReverseProxy` em que realmente o servidor se mostrou disponível. Isto é relevante pois entre o envio desta mensagem e sua recepção pelo `ReverseProxy` o WebServer pode-se desligar. Nesta situação, como a mensagem tem o tempo em que foi enviada, o ReverseProxy sabe sempre o último instante em que o servidor esteve ativo. O último instante em que um servidor está activo será um dos factores que irá influenciar a escolha do melhor servidor. Esta escolha terá por base a informação contida na tabela de monitorização(`MonitorTable`) do `ReverseProxy` e será feita seguindo o algoritmo apresentado no capitulo 4.

Para além deste tipo de mensagem, o `MonitorUDP` é ainda responsável por responder a todas as mensagens periódicas com pedidos de monitorização que o servidor lhe envia. As mensagens enviadas pelo servidor contém o número de sequência da mensagem e são do tipo PROB_REQUEST. Estas mensagens são enviadas de 5 em 5 segundos, sendo o número de sequência incrementado a cada mensagem. A estas mensagens o `MonitorUDP` responde enviando mensagens do tipo PROB_REPLY , com o mesmo número de sequência e o instante em que é enviada. Através desta monitorização activa o `ReverseProxy` consegue determinar informação sobre o RTT e a perda de pacotes, sendo também atualizados os tempos de última vez disponível dos `WebServers`.

3.3 Tabela Monitorização

A tabela de monitorização presente no `ReverserProxy` contêm os seguintes campos: o IP do servidor, o número de conexões TCP do servidor, o instante em que esse servidor se mostrou disponível, o *round-trip-time* e a taxa de pacotes perdidos. O número de pacotes perdidos é monitorizado através do último número de sequência enviado e do último número de sequência recebido. Caso estes não sejam iguais, então ocorreu uma perda de pacote. O RTT é calculado fazendo a diferença entre o instante em que foi enviada e o instante em que esta foi recebida a mensagem. Todos estes valores também se encontram na tabela de monitorização.

4. Implementação

A implementação da arquitetura proposta foi feita com recurso à linguagem Java. Foram criadas classes para a elaboração do `WebServer` e do `ReverseProxy`.

4.1 ReverseProxy

A classe `ReverseProxy` tem como variáveis de instância a tabela de monitorização (`MonitorTable`) e um `ReentrantLock` para o controlo de concorrência nos acessos à tabela.

Esta classe tem como função iniciar duas *threads*: uma responsável pela monitorização e com uma instância da classe `MonitorUDPProxy` e outra encarregue por atender os pedidos dos clientes e com uma instância da classe `TCPMain`.

4.1.1 MonitorUDPProxy

A classe `MonitorUDPProxy` cria um *socket* UDP na porta 5555 e inicia uma *thread* com uma instância da classe `ProxyReciever`, passando-lhe como argumento a tabela de monitorização e o *socket* criado. Esta é responsável por receber as mensagens de disponibilidade e de *PROB_REPLY* vindas dos *Web Servers*.

No caso de se tratar de uma mensagem do tipo *PROB_REPLY*, é verificado o número de sequência da mensagem e, caso corresponda ao último número de sequência enviado, é calculado o RTT e atualizada a entrada da tabela correspondente.

Sempre que é recebida uma mensagem do tipo *DISPONIVEL*, a classe `ProxyReciever` atualiza a tabela de monitorização. Caso exista já uma entrada na tabela para o endereço IP proveniente, é atualizada a respetiva entrada com a informação da última disponibilidade do *Web Server*. Caso não exista ainda, é criada uma `MonitorTableEntry` e inserida na `MonitorTable`. Além disso, é iniciada uma instância de `ProbeRequester`, que é uma *thread* responsável por enviar, periodicamente, os pedidos de *probing* ao *Web Server* e recebe como argumento a entrada da tabela onde guarda os registos de monitorização.

A classe `ProbeRequester` encarrega-se de enviar mensagens do tipo *PROB_REQUEST* para o servidor de *back-end* respetivo. Após o envio da mensagem, é incrementado o número de sequência, a *thread* em execução aguarda 5 segundos e, passado esse tempo, verifica na tabela se foi recebido a resposta do pacote enviado, i.e., se o `ProxyReciever` recebeu alguma mensagem do tipo *PROB_REPLY* com o último número de sequência enviado. Caso não haja registo na tabela de se ter recebido o *PROB_REPLY*, regista-se uma nova perda, caso contrário, regista-se que o pacote foi enviado e recebido com sucesso. Esta classe também é responsável por terminar a ligação com o *Web Server*, caso a última que este esteve disponível tenha sido há mais de 30 segundos. Nesta situação, a entrada da tabela é removida e as threads do `ReverseProxy` dedicadas à comunicação com o WebServer são terminadas.

4.1.2 `TCPMain`

A classe `TCPMain` é a principal responsável pela componente de comunicação entre cliente-servidor do `ReverseProxy`. Possui um `ServerSocket` e um objeto da classe `MonitorTable` como variáveis de instância.

A *thread* iniciada pelo `ReverseProxy` fica a aguardar clientes e, há chegada de um pedido, executa o algoritmo de escolha do melhor servidor de *back-end* (apresentado de seguida) e estabelece uma conexão TCP com esse servidor. Nesse momento, é obtido um *socket* TCP para a comunicação com o cliente e instanciado um outro para a comunicação com o *Web Server*. É incrementado o número de conexões TCP e iniciadas duas novas *threads*: uma responsável por enviar a informação chegada do cliente para o *Web Server* escolhido (`TCPClientListener`) e outra incumbida de enviar ao cliente a resposta ao seu pedido, vinda do *Web Server* (`TCPClientWriter`).

A *thread* `TCPClientListener` lê o pedido do *socket* do cliente, em *bytes*, e escreve esse mesmo pedido no *socket* do *Web Server*. A *thread* `TCPClientWriter` lê a resposta proveniente do servidor de *back-end*, em *bytes*, e escreve-a no *socket* do cliente. Enviada a resposta ao cliente, é removida a conexão e decrementada na tabela de monitorização o número de conexões TCP do *Web Server* correspondente. Deste modo, tem-se o `ReverseProxy` como um simples intermediário.

4.1.3 Algoritmo de escolha

Para a escolha do melhor *Web Server* capaz de responder a um pedido do cliente é executado um algoritmo de escolha, baseado nas informações presentes na tabela de monitorização.

Para cada um dos *Web Servers*, i.e., para cada uma das entradas da tabela (`MonitorTableEntry`), é atribuída uma classificação. No final, o que tiver menor classificação é o escolhido. O cálculo desta classificação é feito tendo por base: o último momento em que o *Web Server* se mostrou disponível; o RTT médio das últimas 10 comunicações; o número de pacotes perdidos nas últimas 10 comunicações; o número de conexões TCP que o respetivo servidor de *back-end* possui no momento.

Os quatro parâmetros usados no cálculo da classificação foram normalizados, por forma a otimizar o algoritmo de escolha. A cada um deles é atribuído um fator, de acordo com a sua importância (a que consideramos). À última vez que esteve disponível e ao RTT foi atribuído um peso igual a 20%, cada um, e ao número de perdas e ao número de conexões TCP um peso de 30%, cada. No final estes valores são somados e é obtida a classificação final.

4.2 `WebServer`

Foi criada uma classe executável `WebServer`, que recebe como argumento o *hostName* do servidor principal, de *front-end*. Esta classe apenas inicia uma *thread*, com uma instância da classe `MonitorUDPServer`.

A classe `MonitorUDPServer` é a classe principal da monitorização de um *WebServer*. Possui como variáveis de instância o *hostName*, recebido como argumento do `WebServer`, e um `DatagramSocket`. Este *socket* UDP é aberto na posta 5555 e serve para permitir a comunicação com o servidor de *front-end*.

Esta classe é responsável por iniciar duas *threads* de monitorização: uma encarregue de enviar anúncios de disponibilidade (`DispAnnouncer`) e outra responsável por responder aos pedidos de *probing* provenientes do servidor principal (`ProbeReplier`).

A *thread* `DispAnnouncer` envia de 3 em 3 segundos um `DatagramPacket` com a mensagem de disponibilidade em *bytes* para o endereço IP do *hostName* já referido.

A *thread* `ProbeReplier` fica à espera de receber *DatagramPackets* do servidor principal e, sempre que recebe algum, constrói um novo com a mensagem de resposta e com o mesmo número de sequência daquele que recebeu. Criada a mensagem de resposta, esta e convertida em *bytes* e enviada para mesmo endereço de onde veio o *PROB_REQUEST*.

5. Testes e resultados

Nesta secção pretende-se demonstrar como as funcionalidades anteriormente referidas funcionam na prática. Para tal, serão apresentados e explicados alguns dos testes efetuados.

5.1 Topologia de teste CORE

Como cenário de teste foi usada uma topologia CORE criada para o efeito, representada na figura 5.1.

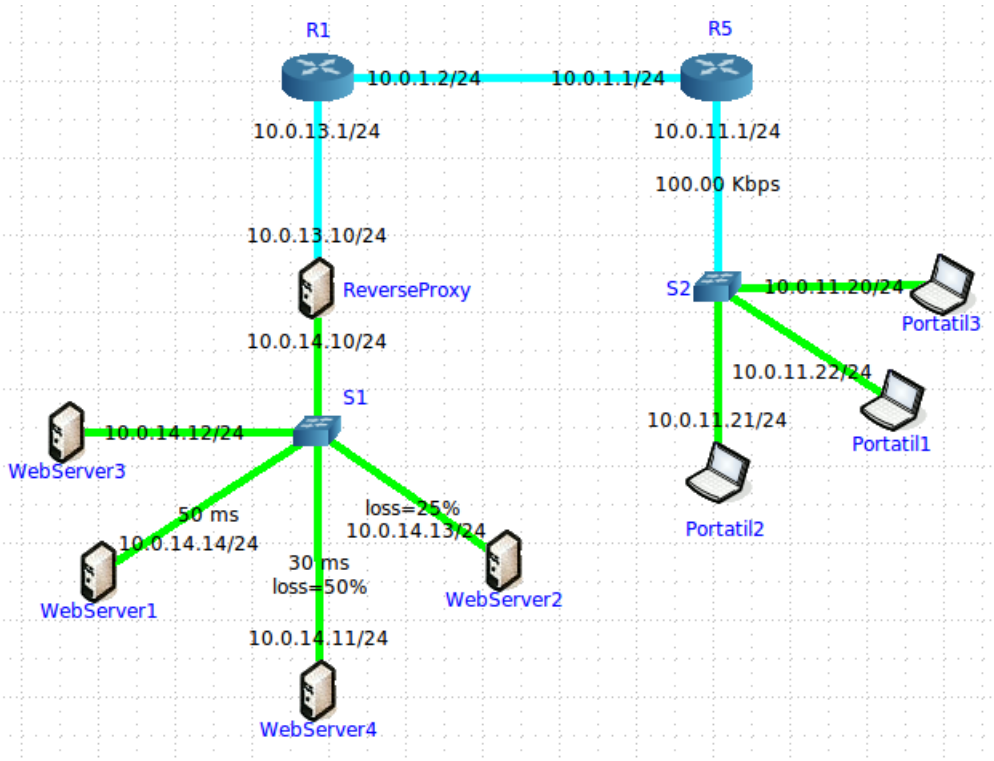

Figura 5.1: Visão geral da topologia de teste criada no CORE

Nesta topologia é possível identificar os vários portáteis que servirão de clientes do *ReverseProxy*. O *ReverseProxy* e os seus *WebServers* também se encontram identificados. De forma a verificar que o cálculo das perdas de pacote e dos RTT estavam a funcionar, foram definidas diferentes características de ligação entre o *ReverseProxy* e cada um dos *WebServers* como se pode ver pela figura 5.2

Além disso, para permitir testes com várias conexões em simultâneo limitou-se a velocidade de ligação aos portáteis como se pode ver na figura 5.3. Isto permite que os pedidos demorem algum tempo a serem satisfeitos, durante o qual se podem realizar mais pedidos.

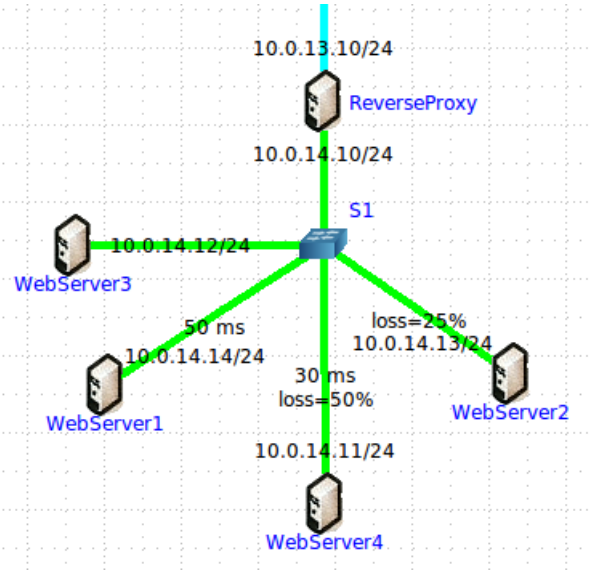

Figura 5.2: Diferentes caracteríticas de ligação entre o ReverseProxy e os WebServers

5.2 Arranque do cenário de teste

Uma vez que tanto o *ReverseProxy* como o *WebServer* foram implementados em Java, foram criados executáveis *.jar* para cada um deles, havendo por isso um *ReverseProxy.jar* que será executado no *ReverseProxy* e um *WebServer.jar* que será executado em cada um dos *WebServer*. Para simular os clientes foram executados comandos como o *wget* nos portáteis da topologia.

5.2.1 Arranque ReverseProxy

O arranque do *ReverseProxy* é feito simplesmente executando o *.jar* respetivo. Para tal foi criado um script *ReverseProxy.sh* com o seguinte:

```
#!/bin/bash
java -jar ~/git-repos/CC-1617/out/jars/ReverseProxy.jar
```

5.2.2 Arranque WebServer

No arranque do WebServer, em primeiro lugar é iniciado o servidor *mini-httpd* sendo depois corrido o *.jar* respetivo. O servidor *mini-httpd* é iniciado na pasta `tp2server` onde serão colocados os ficheiros de teste. De notar que o *WebServer.jar* necessita de receber o endereço do ReverseProxy a que se deve ligar, que neste caso é `10.0.14.10`. O WebServer é por isso iniciado da seguinte forma:

```
#!/bin/bash

mini-httpd -d ~/tp2server/

java -jar ~/git-repos/CC-1617/out/jars/WebServer.jar 10.0.14.10
```

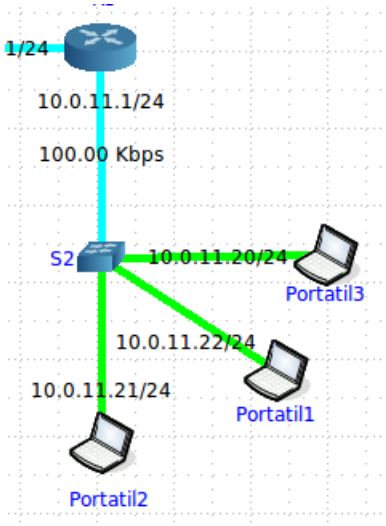

Figura 5.3: Limitação da velocidade de ligação nos portáteis

5.2.3 Arranque Cliente

Os clientes são representados pelos portáteis da topologia. Para se efetuar pedidos http será usado o comando *wget* nos portáteis da topologia. O exemplo apresentado a seguir de que forma se pode pedir ao ReverseProxy um ficheiro "CG.zip". O endereço `10.0.13.10` corresponde ao ReserveProxy (ver figura 5.1). O ficheiro "CG.zip" foi escolhido por ter um tamanho de 19MB. O tamanho do ficheiro em conjunto com a limitação da velocidade de ligação aos clientes (ver figura 5.3) permite que as transferências com o *wget* não sejam instantâneas e haja oportunidade de ter ligações em simultâneo a serem geridas pelo ReverseProxy.

```
#!/bin/bash
wget http://10.0.13.10:80/CG.zip
```

5.3 Testes à monitorização

O primeiro teste básico efetuado à rede foi iniciar um *ReverseProxy* e um *WebServer* e verificar se as PDU's de monitorização estavam a ser trocadas. Para evidenciar a troca de mensagens de monitorização iniciou-se o ReverseProxy e o WebServer3 (que não possui qualquer limitação ou atraso na rede). Os resultados obtidos são mostrados na figura 5.4.

Pela figura 5.4 é possível verificar que os pedidos de anúncio da disponibilidade estão a ser enviados do WebServer para o ReverseProxy (mensagens do tipo `DISPONIVEL`). Além disso, estão a ser enviados pedidos de probing do ReverseProxy para o WebServer (pedidos do tipo `PROB_REQUEST`) aos quais o WebServer está a responder (pedidos do tipo `PROB_REPLY`). Quando o ReverseProxy recebe resposta a um pedido de probing imprime o RTT e a perda de pacotes para aquele servidor. Essa informação também é mostrada na figura. Neste caso o RTT é desprezável (aprox. 8 ms) e a perda de pacotes é nula, o que se justifica visto que se escolheu um WebServer sem nenhuma limitação na ligação. De seguida iremos ver um caso em que essas limitações existem.

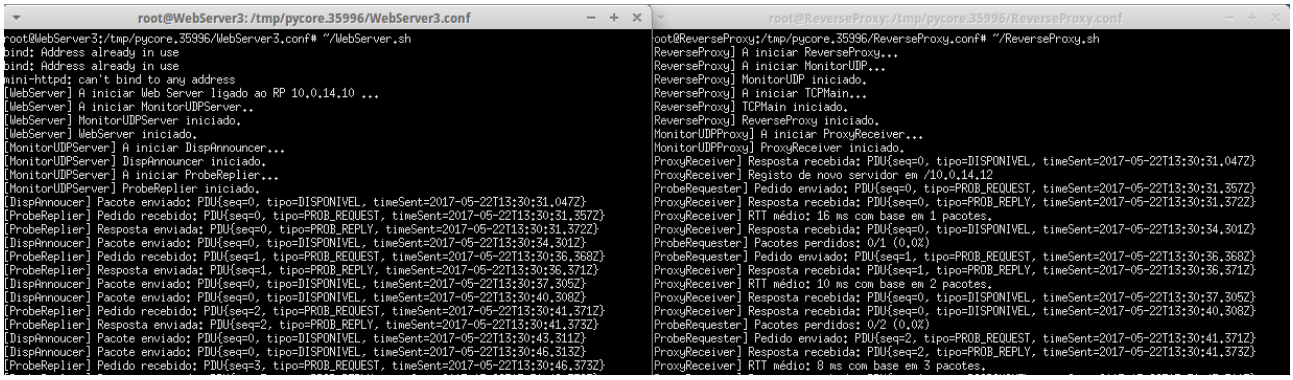

Figura 5.4: Troca de mensagens de monitorização entre o ReverseProxy (lado direito) e o WebServer3 (lado esquerdo)

5.4 RTT e perda de pacotes

Uma vez verificado que as mensagens de monitorização estavam a ser correctamente trocadas, falta testar se o RTT e as perdas de pacotes estão a ser corretamente calculadas e inseridas na tabela. Para tal iniciou-se o ReverseProxy e o WebServer4, que tem um atraso de 30 ms e uma probabilidade de perda de pacotes de 50%. Esperou-se algum tempo para que várias mensagens de monitorização pudessem ser trocadas e o resultado ao fim de 15 pacotes de probing está mostrado na figura 5.5.

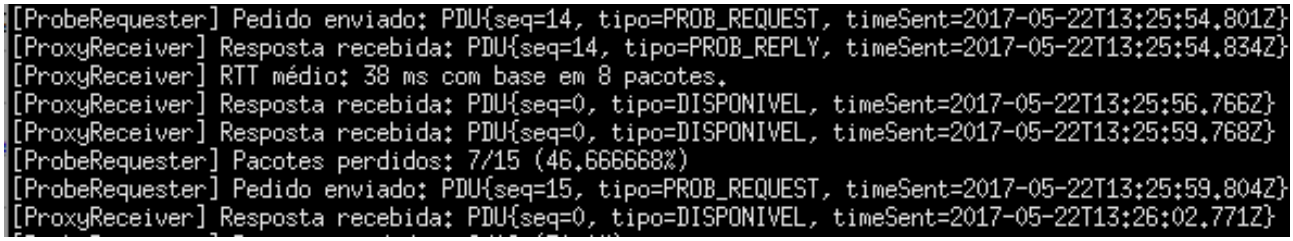

Figura 5.5: Cálculo de RTT e perda de pacotes do ReverseProxy relativamente ao WebServer4

Ao fim dos 15 pedidos de probing, o ReverseProxy registou uma perda 7 pacotes de probing (46.66%) e um RTT médio de 38 ms. Isto é bastante próximo da perda de pacotes de 50% e do atraso de 30 ms definidos na ligação, o que mostra que o ReverseProxy está a conseguir calcular correctamente estes dois parâmetros.

5.5 Fecho de servidores inactivos

Caso o ReverseProxy num período de 30 segundos não receba de um WebServer nenhuma resposta ao pedido de probing ou uma mensagem a anunciar a disponibilidade, assume que o WebServer em questão está *down* e fecha a ligação com esse WebServer. Fechar a ligação implica o término das threads especificas para aquele WebServer e a retirada do servidor da tabela de monitorização.

A figura 5.6 mostra o que está a acontecer no ReverseProxy nesse cenário.

```
[ProxyReceiver] Resposta recebida: PDU{seq=0, tipo=DISPONIVEL, timeSent=2017-05-22T12:53:47.007Z}
[ProxyReceiver] Resposta recebida: PDU{seq=0, tipo=DISPONIVEL, timeSent=2017-05-22T12:53:50.008Z}
[ProbeRequester] Pacotes perdidos: 14/29 (48.27586%)
[ProbeRequester] Pedido enviado: PDU{seq=29, tipo=PROB_REQUEST, timeSent=2017-05-22T12:53:51.069Z}
[ProxyReceiver] Resposta recebida: PDU{seq=29, tipo=PROB_REPLY, timeSent=2017-05-22T12:53:51.106Z}
[ProxyReceiver] RTT médio: 36 ms com base em 16 pacotes.
[ProxyReceiver] Resposta recebida: PDU{seq=0, tipo=DISPONIVEL, timeSent=2017-05-22T12:53:53.010Z}
[ProxyReceiver] Resposta recebida: PDU{seq=0, tipo=DISPONIVEL, timeSent=2017-05-22T12:53:56.011Z}
[ProbeRequester] Pacotes perdidos: 14/30 (46.666668%)
[ProbeRequester] Pedido enviado: PDU{seq=30, tipo=PROB_REQUEST, timeSent=2017-05-22T12:53:56.071Z}
[ProbeRequester] Pacotes perdidos: 15/31 (48.387096%)
[ProbeRequester] Pedido enviado: PDU{seq=31, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:01.073Z}
[ProbeRequester] Pacotes perdidos: 16/32 (50.0%)
[ProbeRequester] Pedido enviado: PDU{seq=32, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:06.074Z}
[ProbeRequester] Pacotes perdidos: 17/33 (51.515152%)
[ProbeRequester] Pedido enviado: PDU{seq=33, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:11.076Z}
[ProbeRequester] Pacotes perdidos: 18/34 (52.941177%)
[ProbeRequester] Pedido enviado: PDU{seq=34, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:16.078Z}
[ProbeRequester] Pacotes perdidos: 19/35 (54.285717%)
[ProbeRequester] Pedido enviado: PDU{seq=35, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:21.079Z}
[ProbeRequester] Pacotes perdidos: 20/36 (55.555557%)
[ProbeRequester] Pedido enviado: PDU{seq=36, tipo=PROB_REQUEST, timeSent=2017-05-22T12:54:26.081Z}
[ProbeRequester] Pacotes perdidos: 21/37 (56.75676%)
[ProbeRequester] A terminar ligação a /10.0.14.11 por inactividade.
```

Figura 5.6: Ligação a WebServer fechada pelo ReverseProxy devido a inactividade

A troca de mensagens de monitorização decorre normalmente até que o servidor é fechado, momento a partir do qual se vêm várias mensagens `PROBE_REQUEST` que não obtêm resposta. Ao fim de 30 segundos o ReverseProxy decide fechar a ligação aquele WebServer.

5.6 Clientes e escolha de servidor

Para testar o algoritmo de escolha dos servidores, iniciou-se o ReverseProxy e os 4 WebServers da topologia. De seguida, efectuou-se em simultâneo nos 3 portáteis da topologia um pedido de *wget* ao ficheiro “CG.zip”. Sempre que recebe e aceita um pedido, o ReverseProxy escreve num ficheiro “server-choice-log.txt” o estado da tabela de monitorização no momento em o pedido foi recebido e qual a decisão tomada em função dessa tabela.

Ao receber o primeiro pedido o ReverseProxy reportou a informação mostrada na figura 5.7.

```
=== NOVO PEDIDO ===
Pontuacao para /10.0.14.11 = 0.31183 disp: 0.00049 (98 ms), rtt: 0.01134 (34 ms), perdas: 0.3 (5), con: 0 (0)
Pontuacao para /10.0.14.12 = 0.00837 disp: 0.00738 (1474 ms), rtt: 0.00101 (3 ms), perdas: 0 (0), con: 0 (0)
Pontuacao para /10.0.14.13 = 0.18796 disp: 0.00662 (1324 ms), rtt: 0.00134 (4 ms), perdas: 0.18 (3), con: 0 (0)
Pontuacao para /10.0.14.14 = 0.02008 disp: 0.00241 (481 ms), rtt: 0.01767 (53 ms), perdas: 0 (0), con: 0 (0)
 ---> Escolhido melhor server: /10.0.14.12 com pontuacao 0.00837
```

Figura 5.7: Tabela de monitorização e escolha de servidor para 1º pedido

Visto que não possui nenhuma limitação na ligação, sem surpresa, o servidor escolhido para atender o primeiro pedido foi o WebServer3 (`10.0.14.12`).

Ao receber o segundo pedido o ReverseProxy reportou a informação mostrada na figura 5.8.

```
=== NOVO PEDIDO ===
Pontuacao para /10.0.14.11 = 0.31565 disp: 0.00432 (863 ms), rtt: 0.01134 (34 ms), perdas: 0.3 (5), con: 0 (0)
Pontuacao para /10.0.14.12 = 0.10106 disp: 0.00006 (11 ms), rtt: 0.00101 (3 ms), perdas: 0 (0), con: 0.1 (1)
Pontuacao para /10.0.14.13 = 0.19155 disp: 0.01021 (2042 ms), rtt: 0.00134 (4 ms), perdas: 0.18 (3), con: 0 (0)
Pontuacao para /10.0.14.14 = 0.02366 disp: 0.006 (1198 ms), rtt: 0.01767 (53 ms), perdas: 0 (0), con: 0 (0)
 ---> Escolhido melhor server: /10.0.14.14 com pontuacao 0.023656666666666666
```

Figura 5.8: Tabela de monitorização e escolha de servidor para 2º pedido

Neste segundo pedido podemos ver que o WebServer3, que tinha sido escolhido para o 1º pedido, agora possui uma conexão activa, o que afectou a sua pontuação. Isto fez com que o WebServer1 (`10.0.14.14`) fosse o escolhido para este 2º pedido.

Ao receber o terceiro pedido o ReverseProxy reportou a informação mostrada na figura 5.9.

```
=== NOVO PEDIDO ===
Pontuacao para /10.0.14.11 = 0.32032 disp: 0.00899 (1797 ms), rtt: 0.01134 (34 ms), perdas: 0.3 (5), con: 0 (0)
Pontuacao para /10.0.14.12 = 0.10573 disp: 0.00473 (945 ms), rtt: 0.00101 (3 ms), perdas: 0 (0), con: 0.1 (1)
Pontuacao para /10.0.14.13 = 0.19622 disp: 0.01488 (2976 ms), rtt: 0.00134 (4 ms), perdas: 0.18 (3), con: 0 (0)
Pontuacao para /10.0.14.14 = 0.12834 disp: 0.01067 (2133 ms), rtt: 0.01767 (53 ms), perdas: 0 (0), con: 0.1 (1)
 ---> Escolhido melhor server: /10.0.14.12 com pontuacao 0.10572499999999999
```

Figura 5.9: Tabela de monitorização e escolha de servidor para 3º pedido

Apesar de já possuir uma ligação activa (do 1º pedido), o WebServer3 (`10.0.14.12`) voltou a ser escolhido para atender o 3º pedido. Pela pontuação vê-se que WebServer1 (`10.0.14.14`) esteve também perto de ser escolhido. O facto da perda de pacotes ser o factor mais importante da fórmula da pontuação, colocou em grande desvantagem o WebServer3 (`10.0.14.11`) e sobretudo o WebServer4 (`10.0.14.11`), uma vez que ambos os servidores têm ligações com perdas.

6. Conclusão

Os objetivos propostos para o trabalho foram cumpridos. Foi conseguida a implementação de um ReverseProxy capaz de monitorizar via UDP WebServers que lhe estejam associados e de responder via TCP a pedidos de recursos dos WebServers por parte de clientes.

Esta implementação foi feita em Java e envolveu o uso de sockets para a comunicação dos clientes com o ReverseProxy e deste com os WebServers. Para que o ReverseProxy fosse capaz de gerir vários WebServers e vários clientes a fazer pedidos foi necessário o uso de threads. De forma genérica, pode-se dizer que cada thread é responsável por uma determinada tarefa para cada cliente/WebServer. Neste contexto, as técnicas de controlo de concorrência também se revelaram importantes, sobretudo porque o ReverseProxy contém um recurso, neste caso a tabela de monitorização, que pode ser acedido pelas várias threads.

Como possíveis melhorias ao trabalho, sugere-se a melhor organização dos *logs* do servidor WebServers e a possibilidade de interagir com o WebServer ou com o ReverseProxy através de comandos. No estado atual do trabalho, sempre que algo de relevante acontece no WebServer ou no ReverseProxy, é impresso no ecrã uma mensagem informativa do que aconteceu. Estas mensagens podem ser impressas por várias threads que estão responsáveis por tarefas diferentes. Quando o volume de WebServers e de pedidos de clientes aumenta, estas mensagens rapidamente ficam confusas e impercetíveis, o que contraria o objetivo de as ter. Para tal sugere-se que estas mensagens em vez de serem escritas para o ecrã, sejam escritas para ficheiros separados, ficando desta forma ordenadas "por assunto" e facilitando a navegação pelas mesmas. Como forma se poder continuar a ter alguma informação dada pelas mensagens no ecrã, sugere-se a implementação de comandos no WebServer e no ReverseProxy que permitam aceder a essa informação. Por exemplo, no ReverseProxy seria interessante ter um comando `table` que mostrasse a tabela de monitorização a pedido do utilizador. Estes comandos poderiam não só ser usados para obter informação como poderiam ser usados para o próprio utilizador poder interagir diretamente com o WebServer / ReverseProxy. Por exemplo, o WebServer poderia ter um comando `exit` que permitisse a um administrador fechar "civilizadamente" um WebServer. Ao ser reconhecido este comando, o WebServer poderia enviar uma mensagem ao ReverseProxy a indicar que iria fechar, evitando os 30 segundos de espera e as mensagens de probing desnecessárias durante esse tempo por parte do ReverseProxy.