Reverse Proxy

João Pedro Ferreira Vieira
A78468@alunos.uminho.pt

José Carlos Lima Martins
A78821@alunos.uminho.pt

Miguel Miranda Quaresma
A77049@alunos.uminho.pt

Universidade do Minho
Comunicação por Computadores

10 de Maio de 2018

Resumo

Após a implementação de um serviço de distribuição de carga com servidor proxy, procedemos à documentação do mesmo, de modo a ser perceptível o que foi desenvolvido. É apresentada a arquitetura da solução bem como os formatos dos pacotes. É de seguida explicada a implementação para por fim apresentar os testes e os resultados obtidos.

1 Introdução

O trabalho desenvolvido tem como objetivo a implementação de um serviço de distribuição de carga com servidor proxy invertido usado quando há a necessidade de mais do que um servidor a atender vários clientes devido ao grande volume dos mesmos e pretende-se manter um único endereço IP público.

2 Arquitetura da solução

Pretende-se implementar um serviço de balanceamento de carga com servidor proxy invertido, no qual existe um servidor com um endereço IP público que atende todos os clientes, desviando-os para um dos servidores HTTP de back-end disponíveis. Este servidor é chamado de Reverse Proxy. Uma coisa que é importante referir é que no nosso trabalho as conexões dos clientes não sao desviadas mas é sim criado um "tunel" no Reverse Proxy entre o servidor HTTP de back-end escolhido e o cliente, de modo a simplificar a implementação, contudo, isto cria um ponto de falha grave no Reverse Proxy. Este Reverse Proxy pode ser dividido em 3 partes: MonitorUDP, tabela de estado e o "Reverse Proxy".

O MonitorUDP envia em Multicast um datagrama UDP de 3 em 3 segundos para todos os agenteUDP's (um por cada servidor HTTP back-end) de modo

a que cada agenteUDP responda em Unicast com o estado do servidor. O MonitorUDP ao receber um datagrama UDP de um dos agenteUDP's calcula o RTT(Round Trip Time), obtém o IP de origem e porta de origem do datagrama e com toda esta informação atualiza a tabela de estado referente ao servidor HTTP back-end do qual foi recebido o datagrama.

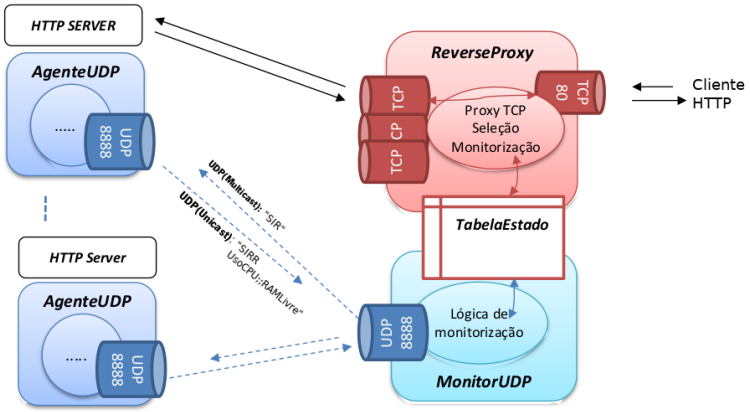
A tabela de estado possui em cada linha informação referente a um servidor HTTP back-end, sendo que em cada linha a informação presente é organizada da seguinte maneira:

```
IP;;Porta;;CPU_usado;;RAM_livre;;RTT;;BW(bandwidth)
```

em que o 3º parâmetro (CPU_usado) é uma percentagem, a RAM_livre é um tamanho em bytes, o RTT representa um tempo em milisegundos, e a BW (bandwidth) é um tamanho em Kbytes por segundo (Kbps).

Quanto ao "Reverse Proxy", quando recebe a conexão de um cliente tem que escolher da tabela de estado um servidor HTTP (usando um algoritmo definido por nós), criar uma ligação TCP entre o mesmo e o cliente e ficar encarregue de ir atualizando na tabela de estado o valor da bandwidth.

Esta arquitetura encontra-se ilustrada na seguinte figura:



3 Especificação do protocolo UDP

3.1 Formato das mensagens protocolares (PDU)

Existem dois tipos de mensagens realizadas entre os UDPAgent's e o UDP-Monitor, a mensagem enviada pelo UDPMonitor em Multicast para todos os UDPAgent's, e a mensagem de cada UDPAgent quando recebe a mensagem do UDPMonitor para o mesmo em Unicast.

Quanto à mensagem em Multicast, o formato é simples visto ser apenas necessário informar que a mensagem recebida foi enviada pelo UDPMonitor,

como tal a mensagem enviada é apenas "SIR". Já a mensagem em Unicast, tem de possuir toda a informação a enviar desde o servidor (UDPAgent) para o UDPMonitor e como tal o formato da mensagem é a seguinte:

```
SIRR
percentagem_de_uso_do_CPU;;memoria_RAM_livre
```

Convém referir que os dois ;'s (pontos e vírgulas) são usados de modo a separar corretamente os diferentes dados e que SIRR tem como significado "System/Server Information Request Response".

3.2 Interações

Sendo assim, são estabelecidas várias interações entre os intervenientes deste *Reverse Proxy*, nomeadamente entre:

- **UDPAgent e UDPMonitor**
  O UDPMonitor envia em *multicast* para todos os UDPAgent's no grupo a mensagem "SIR", estes respondem à mesma em *unicast* com os valores do servidor em que estão a correr, de forma a poder ser atualizada a sua informação na Tabela de Estados do programa. É importante referir que a autenticidade e integridade das mensagens trocadas entre estes dois intervenientes é garantida.

- **UDPMonitor e StateTable**
  Sempre que o UDPMonitor recebe informações dos servidores através dos UDPAgent's cabe-lhe a função de atualizar os valores destes servidores na Tabela de Estados caso estes já sejam pertencentes ou grupo, ou simplesmente adicionar um novo servidor com a sua respetiva informação à tabela em causa.

- **Cliente e ReverseProxy**
  Quando um cliente requisita uma consulta ao ReverseProxy este escolhe o melhor servidor no devido momento tendo em conta os valores dos mesmos na Tabela de Estados (cálculo através do algoritmo) e cria um "túnel"entre esse mesmo servidor e o cliente de forma a estabelecer a conexão necessária.

- **ReverseProxy e StateTable**
  Cabe ao ReverseProxy calcular a *bandwidth* sempre que um cliente faça um *download* através de um dos servidores disponíveis. Com isto, cabe ao ReverseProxy atualizar o valor da *bandwidth* desse mesmo servidor na Tabela de Estados tendo em conta o valor calculado.

4 Implementação

O trabalho foi desenvolvido em Java e como tal decidimos dividir o código pelas seguintes classes:

- StateTable: Representa a tabela de estados
- UDPMonitor: Envia datagramas UDP de 3 em 3 segundos para o canal multicast
- UDPAgent: Responde em unicast ao MonitorUDP com o estado do servidor HTTP a que está associado
- ListenUDPAgents (Thread do UDPMonitor): Recolhe os datagramas enviados pelos UDPAgent's e atualiza com estes diagramas a tabela de estados (StateTable)
- Timer: Usado pelo UDPMonitor e pelo ListenUDPAgents de modo a calcular o RTT das ligações com os servidores
- ReverseProxy: O Reverse Proxy funciona como um servidor TCP recebendo pedidos na porta 80 dos clientes
- Connection (Thread do ReverseProxy): criada para tratar de cada cliente, ou seja cada cliente possui uma Thread Connection no ReverseProxy sendo que esta Thread trata de escolher o servidor HTTP da tabela de estados, inicia a Thread ListenFromClient e fica à escuta do servidor HTTP e envia os dados para o cliente
- ListenFromClient (Thread da Connection): Envia os dados que recebe do cliente para o servidor HTTP

Tanto a classe Timer bem como StateTable devem possuir lock's de modo a impedir que Threads/Processos diferentes tenham acesso simultaneamente aos dados, evitando dead lock's e que os dados sejam "corrompidos".

Em relação às bibliotecas usadas, foi necessário o uso de uma biblioteca externa às do Java. Esta biblioteca foi o Sigar (System Information Gatherer and Reporter) da Hyperic, que nos permite obter os dados como percentagem de uso de CPU e de memória livre do Sistema no todo (máquina com UDPAgent e servidor HTTP a correr) e não apenas o que a máquina virtual do Java disponibilizou para a execução do UDPAgent. Para além disso funciona em vários sistemas operativos para além do GNU/Linux. É, contudo, importante salientar que nos nossos testes efetuados, como os mesmos são realizados numa mesma máquina, os dados obtidos pelos vários UDPAgent's é o mesmo. Desta biblioteca foram usado as seguintes funcionalidades:

- org.hyperic.sigar.Sigar: necessário de modo a poder usar as funcionalidades seguintes
- org.hyperic.sigar.Mem: permite a obtenção de dados referentes à memória RAM
- org.hyperic.sigar.Cpu: permite a obtenção de dados referentes ao CPU

Já em relação às bibliotecas predefinidas do Java as mais importantes a referir são as que permitem a conexão entre os diferentes componentes seja por TCP ou UDP e a que nos permite garantir a integridade bem como autenticação dos pacotes, as mesmas são as seguintes:

- java.net.DatagramPacket: permite a criação de datagramas UDP
- java.net.DatagramSocket: permite a criação de sockets UDP
- javax.crypto.Mac: permite o uso do algoritmo HMAC (hash-based message authentication code) de modo a garantir integridade e autenticação dos pacotes transferidos entre UDPAgent's e UDPMonitor
- java.net.InetAddress: representa um endereço IP
- javax.crypto.spec.SecretKeySpec: permite a criação de uma chave a partir de um conjunto de bytes (ou a partir de String com a conversão da mesma para bytes) com um determinado algoritmo à escolha
- java.net.ServerSocket: permite a criação de um servidor TCP que pode aceitar conexões de clientes através de uma porta à escolha
- java.net.Socket: permite a criação de sockets TCP
- java.net.MulticastSocket: permite a criação de um grupo Multicast

Quanto ao algoritmo de escolha do servidor HTTP por parte do Reverse Proxy o mesmo tem em conta os parâmetros memória livre, percentagem de CPU usado, RTT e bandwidth. Cada um dos parâmetros tem o mesmo peso ou seja 25%(1/4=0.25). Contudo, como todos os valores não são percentagens é necessário garantir isso mesmo de modo a haver uma escolha justa e correta. Sendo assim o RTT é dividido por 3000 ms (3s, o tempo entre datagramas enviados pelo UDPMonitor). Já em relação à memória livre e ao bandwidth é guardado em duas variáveis o maior valor que já apareceu de cada um (maxRam e maxBW). Depois divide-se a memoria livre e a bandwidth por estes valores garantindo assim que serão sempre entre 0 e 1. Como tal é aplicado a cada linha da tabela a seguinte fórmula:

```
res = 0.25 * percentagem_de_uso_do_CPU - 0.25 * memoria_RAM_livre / maxRAM
+ 0.25 * RTT / 3000 + 0.25 * BW / maxBW;
```

sendo depois escolhido o servidor com o menor valor. O valor da memória livre é subtraido devido a ser algo "bom"e não "mau"e como se pretende calcular o minimo.

Para garantir a autenticidade e a integridade nas mensagens trocadas entre o UDPMonitor e o UDPAgent decidimos implementar uma chave/password conhecida tanto pelo Monitor como pelos Agent's, esta password foi decidida por nós e tem o valor de "`abcdfasdgasefdgsdp`". A partir daqui e supondo que `msg` é o conteúdo que é necessário ser enviado do Agent para o Monitor, acrescentamos no início desta mesma mensagem uma `hash` produzida tendo em conta

a key apresentada em cima. Desta forma, o pacote enviado pelo UDPAgent ao UDPMonitor é "`hash` (32 bytes) + `msg`". De seguida, quando este mesmo pacote é recebido pelo Monitor, este extrái o mesmo para um conjunto de bytes, e separa a `hash` da `msg`. Após, isto, calcula uma nova `hash` da `msg` tendo em conta a key que tanto este como os Agent's conhecem, se tudo correr como planeado, o valor desta `hash` calculada é igual ao valor da `hash` recebida. Desta forma e através deste algoritmo conseguimos garantir simultaneamente a autenticidade e a integridade das mensagens que são trocadas entre o UDPMonitor e os UDPAgent's. Fica assim explicitado o HMAC.

Em relação ao cálculo da bandwidth, o mesmo apresentou algumas dificuldades, contudo obteve-se o resultado pretendido adicionando o bandwidth (BW) atual da ligação seja entre cliente-ReverseProxy seja entre servidor HTTP-ReverseProxy à existente na tabela de estados a cada conjunto de 1024 bytes. Porém esta abordagem apresentava um problema, visto que, a bandwidth iria aumentar indefinidamente, algo não real. Como forma a resolver este problema, é guardado valor da bandwidth anterior (prevBW) e, na altura da adição, em vez de se adicionar BW adiciona-se (BW-prevBW) resolvendo assim o problema do aumento indefinido. Para além disso, de modo a que quando uma determinada ligação caia esta nao mantenha o valor da bandwidth na tabela (visto a mesma já nao influenciar), o que se faz é subtrair -BW ao valor existente na tabela de estados. Apesar de tudo, a nossa abordagem ainda apresentava um problema, visto que a maior parte das vezes a bandwidth era igual a 0 e isso devia-se ao pequeno intervalo de tempo e pequeno número de bytes em que era calculado a bandwidth. Sendo assim, em vez de ser a cada ciclo de 1024 bytes é calculado de 20 em 20 ciclos de 1024 bytes. É importante referir que a fórmula usada para calcular a bandwidth é a seguinte:

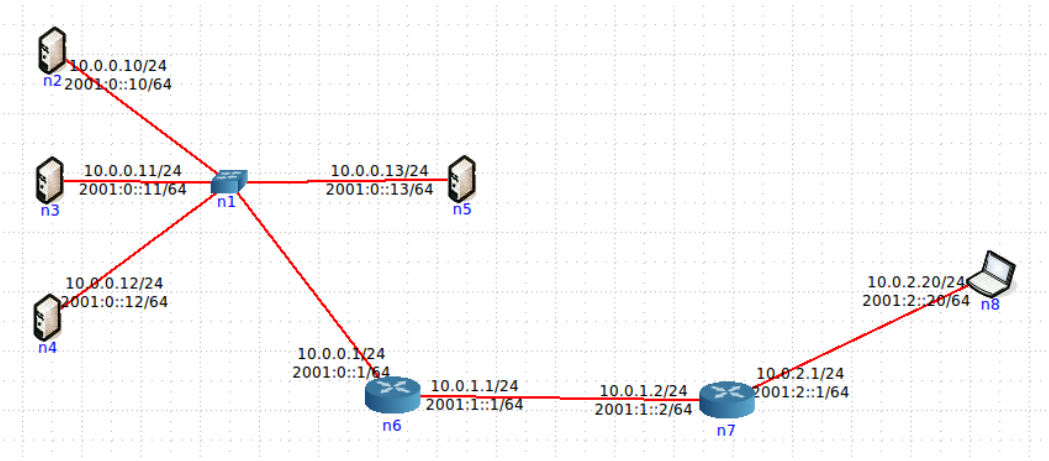
```
time (em s) = (timeInicio (em ms) - timeFim (em ms)) / 1000
BW (em Kbps) = (nRT (número de bytes lidos)/1024) / time
```

Por fim, caso numa determinada ligação o número de ciclos seja menor que 20 ou não seja multiplo de 20 o que é feito é o mesmo, ou seja é aplicado a fórmula que se apresenta acima.

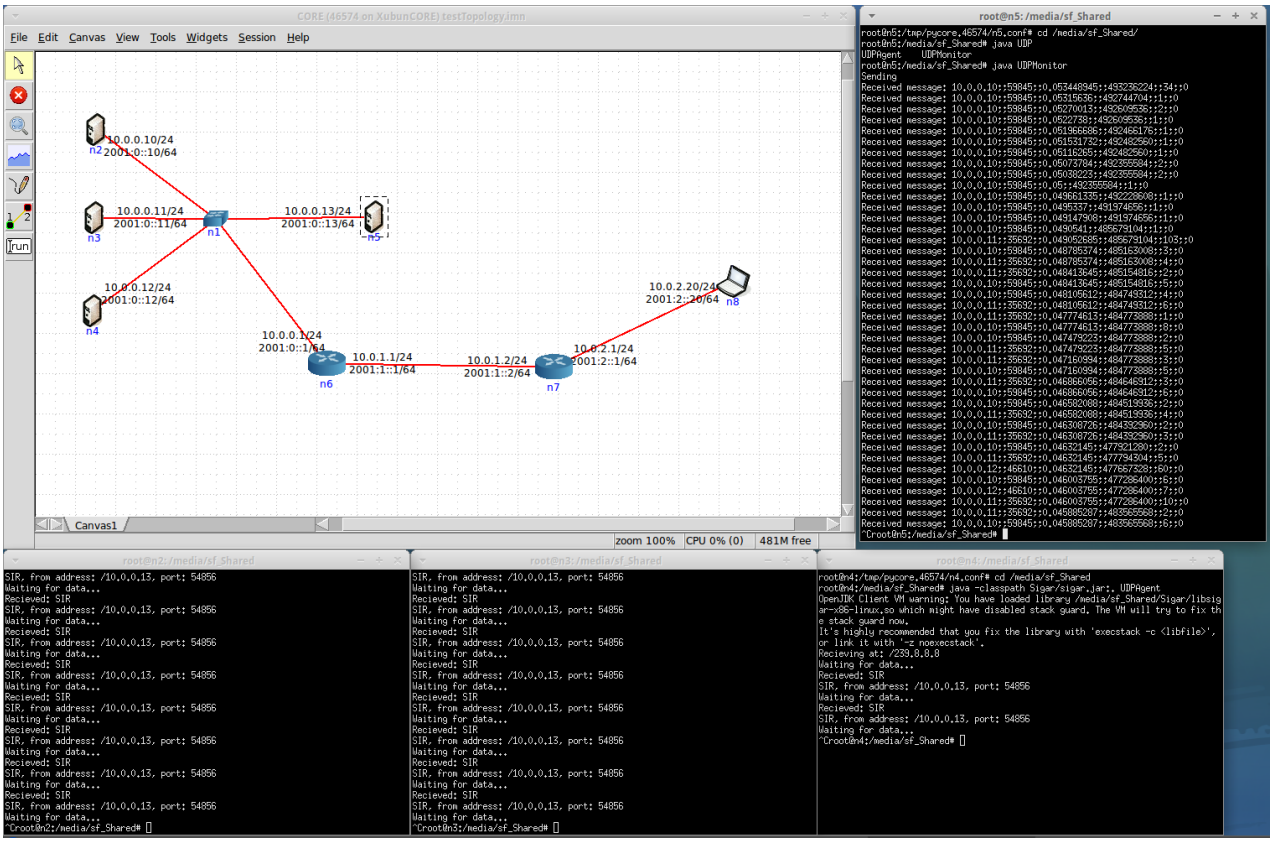
TODO: detalhes, parâmetros, bibliotecas de funções, etc.

5 Testes e resultados

De modo a testar o que foi desenvolvido foi usado o Core (Common Open Research Emulator), um emulador de redes numa máquina, sendo que o mesmo foi usado na máquina virtual fornecida pelos docentes. A topologia usada consistiu no seguinte:



Numa primeira fase foi testada a comunicação UDP entre os UDPAgent's e o UDPMonitor através da execução de 3 UDPAgent's nas máquinas n2,n3 e n4 e o UDPMonitor na máquina n5 e observar o debug como presente na imagem seguinte:



De modo a poder executar é necessário compilar o código:

```
$ javac UDPMonitor.java StateTable.java ReverseProxy.java
$ javac -classpath Sigar/sigar.jar UDPAgent.java
```

Para executar o Reverse Proxy bem como o UDPMonitor é necessário executar na máquina:

```
$ java UDPMonitor
```

Já o UDPAgent nas outras máquinas deve ser executado da seguinte maneira:

```
$ java -classpath Sigar/sigar.jar:. UDPAgent
```

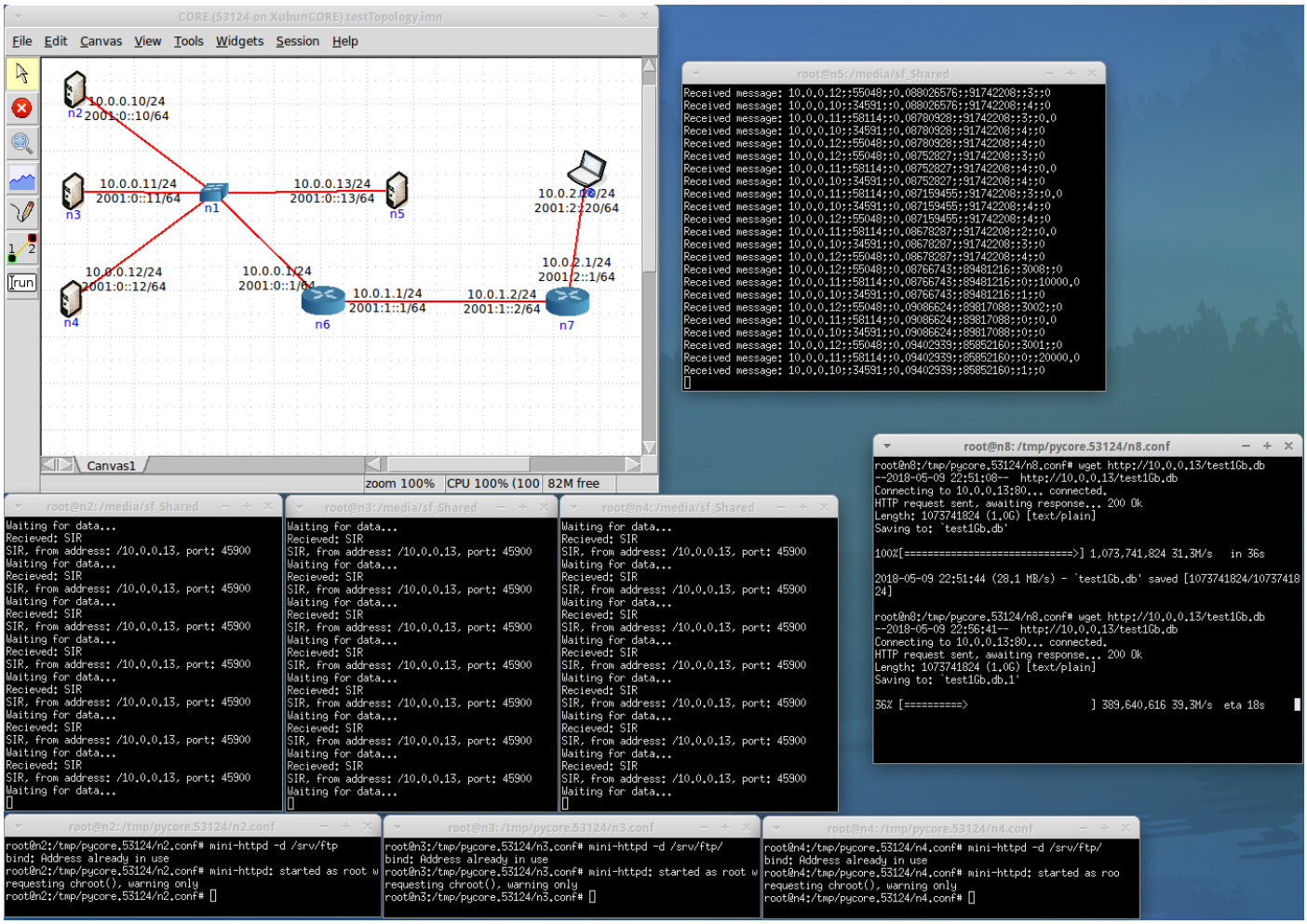
Já numa segunda fase foi testado a escolha do servidor HTTP por parte do Reverse Proxy bem como se a criação do canal entre o mesmo e o cliente era feito de forma transparente. Como tal foi executado em cada máquina onde corre o UDPAgent (máquina n2, n3 e n4) o seguinte comando:

```
$ mini-httpd -d /srv/ftp/
```

criando assim um servidor HTTP em cada uma, em que o mesmo possui dois ficheiros dos quais se pode realizar download. Para além disso é colocado a executar na máquina n5 o Reverse Proxy e o UDPMonitor. Após isso foi feito um pedido wget a partir do cliente com o endereço IP 10.0.2.20, por exemplo:

```
$ wget http://10.0.0.13/teste1Gb.db
```

ao servidor Reverse Proxy (endereço IP: 10.0.0.13). Depois basta verificar se o ficheiro teste1Gb.db está presente no cliente e em caso positivo pode-se concluir que a operação funcionou corretamente. Ilustra-se na imagem seguinte o teste realizado:



6 Conclusões e trabalho futuro

Apesar de o trabalho cumprir os requisitos propostos poderia ainda ser melhorado, no sentido em que em vez do Reverse Proxy realizar a ligação entre o servidor back-end e o cliente, seria em alternativa enviado ao cliente o endereço do servidor back-end de modo ao mesmo ligar-se a ele, deixando assim dos dados passarem pelo Reverse Proxy, e evitando que se estabeleça aqui um ponto de falha.

7 Referências

Imagem na secção Arquitetura da Solução: Enunciado do Trabalho
Sigar Download: https://sourceforge.net/projects/sigar/
Sigar API: http://cpansearch.perl.org/src/DOUGM/hyperic-sigar-1.6.3-src/docs/javadoc/index.html