Trabalho Prático Nº2 – Proxy TCP reverso com monitorização proativa

Alexandre Silva, Hugo Carvalho e Luís Lima

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

{a73674, a74219, a74260}@alunos.uminho.pt

Abstract. O presente relatório foi produzido no âmbito da Unidade Curricular de Comunicações por Computador do 3º ano do Mestrado Integrado em Engenharia Informática. Este consiste no desenho e implementação de um serviço de proxy reverso TCP.

Keywords: ReverseProxy, server, comunicação, protocolo, udp, tcp, socket.

1 Introdução

Um serviço de Proxy Reverso consiste num servidor *front-end* que tem como principal objetivo receber conexões de cliente e desviá-las para servidores *back-end* disponíveis. Assim, este servidor tem um nome e um endereço IP únicos e conhecidos e possui um único ponto de entrada para cada cliente.

O sistema descrito neste relatório foi dividido em duas fases: criação de um protocolo de monitorização da *pool* de servidores e implementação do Proxy TCP reverso.

Na primeira parte foi necessário desenhar e implementar um protocolo sobre UDP que permitisse criar e atualizar uma tabela com informações recolhidas pelo servidor como IP do servidor, *round-trip time (RTT)* estimado entre o *front-end* e o servidor, taxa de pacotes perdidos e número de conexões TCP do servidor. Com este objetivo foi desenvolvido o agente "MonitorUDP".

Na segunda parte foi desenvolvido o servidor que atende pedidos TCP na porta 80, aceita a conexão e decide qual o melhor servidor disponível naquele instante consultando a tabela de estado. De seguida, abre uma conexão TCP com a porta 80 desse servidor e funciona como intermediário entre estas duas conexões.

2 Arquitetura da Solução Implementada

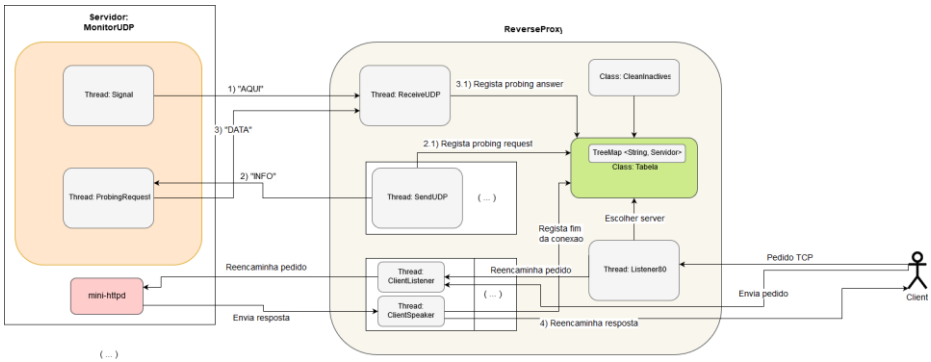

Figura 1 Arquitetura

Este trabalho foi desenvolvido em duas fases:

Na primeira fase deste trabalho é pedido que exista um agente de monotorização implementado nos servidores (MonitorUDP) e que o servidor proxy reverso seja capaz de interpretar e guardar as informações necessárias relativas a cada servidor ativo. Posto isto e para darmos solução ao problema em questão resolvemos criar duas *threads* para realizar esta monotorização. A *thread* Signal é responsável por enviar periodicamente uma mensagem a sinalizar a presença daquele servidor como disponível e a *thread* ProbingRequest que está à espera de um pedido de informações por parte do proxy reverso e quando chega esse pedido imediatamente envia uma resposta.

A parte do proxy reverso é ligeiramente mais complexa devido a ter que atender e gerir as mensagens de monotorização de todos os servidores, bem como analisar e guardar as informações pertinentes relativas a estes servidores.

Assim sendo, existem dois tipos de *threads* para responsáveis pela comunicação com os servidores, uma para receber e outra para enviar mensagens: ReceiveUDP e SendUDP. A primeira é uma *thread* que está sempre ativa e à espera de pacotes, sendo responsável por analisar estes pacotes e tomar uma ação. Se for uma mensagem a sinalizar a presença de um servidor é criada uma *thread* SendUDP que vai enviar um *probing request* a esse servidor. Por outro lado, caso a mensagem seja a resposta ao *probing request* apenas é realizado o armazenamento dessa informação.

Para guardar a informação de todos os servidores ativos implementámos uma classe denominada Tabela. Esta classe possui um *TreeMap* com objetos do tipo Servidor. Estes objetos possuem toda a informação relativa a um servidor.

Como os servidores podem deixar de estarem disponíveis, foi criada uma *thread* CleanInactives que periodicamente elimina da Tabela todos os servidores que não manifestaram a sua presença à mais de um certo tempo.

Na segunda fase deste trabalho é necessário que o proxy reverso atenda os pedidos TCP na porta 80 e que os reencaminhe para um servidor disponível com base num algoritmo de seleção.

Assim sendo, temos uma *thread* Listener80 que atende conexões, seleciona um servidor e cria duas *threads* que serão responsáveis pela realização da ponte de comunicação entre o servidor escolhido e o cliente (ClientListener e ClientSpeaker). A *thread* ClientSpeaker, além de ser responsável por encaminhar para o cliente todas as mensagens vindas do servidor é também responsável por atualizar a tabela quando a conexão termina.

3 Especificação do Protocolo de Monitorização

Para a realização da comunicação entre servidores e proxy reverso, tem que existir um protocolo bem definido.

Neste caso, cada servidor envia periodicamente uma mensagem ao proxy reverso com a mensagem "AQUI". O proxy reverso, quando recebe esta mensagem, imediatamente envia uma *probing request*, ou seja, uma mensagem com o texto "INFO". Deste modo o servidor sabe que o proxy reverso quer uma resposta com dados sobre o servidor. Após a receção desta mensagem o servidor envia uma resposta ao *probing* com a mensagem "DATA". A razão desta mensagem apenas conter a string "DATA" e não dados específicos do servidor em questão é porque todas as informações que o proxy reverso necessita sobre o servidor podem ser adquiridas nos atributos do *DatagramPacket.*

Estas três interações mencionadas em cima fecham assim um ciclo normal de monotorização entre um servidor e o proxy reverso. Na figura 1 é apresentado graficamente estas interações.

4 Implementação

Neste tópico vamos falar sobre alguns parâmetros bem como foram implementadas algumas funcionalidades.

Para a realização periódica de envio de mensagens a sinalizar a presença dos servidores, o grupo escolheu 10 segundos. Por outro lado, a *thread* CleanInactives "inspeciona" a tabela de 20 em 20 segundos e elimina servidores inativos há mais de 30 segundos. Numa situação real, o grupo acha que estes valores são muito pequenos e gera uma grande troca de mensagens de monotorização, contudo, como este é um caso de estudo, o grupo optou por estes valores baixos para podermos observar todas estas ações a funcionarem.

Para o cálculo do rtt (round-trip time), sempre que o proxy reverso envia um *probing request* ao servidor, regista o tempo e guarda-o na tabela. De seguida, quando recebe a resposta do servidor, volta a registar o tempo, calcula a diferença e posteriormente a média com os rtt anteriores.

Sempre que o servidor envia duas mensagens a sinalizar a sua presença e entre estas mensagens não foi registada a chegada de nenhuma resposta ao *probing request*, o proxy reverso vai entender que um pacote foi perdido.

5 Testes e Resultados

Ao longo e no final da implementação foram realizados vários testes para verificar o correto funcionamento do sistema.

Para os testes, o grupo decidiu aceitar as sugestões do professor e usou o emulador CORE, bem como servidor web mini-httpd.

6 Conclusões e Trabalho Futuro

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos da unidade curricular de Comunicações por Computador. Concluído este trabalho, acreditamos ter desenvolvido um serviço de proxy reverso TCP que responde a todos os requisitos propostos.

Como trabalho futuro alguns aspetos podiam ser revistos e melhorados, como por exemplo a implementação de um outro algoritmo na escolha do melhor servidor disponível naquele instante para o atendimento de pedidos.

Foram aplicados conhecimentos que adquirimos ao longo das aulas, sendo estes desenvolvidos à medida que o trabalho ia sendo realizado, uma vez que foi necessário resolver um conjunto de novas situações e criar mecanismos de raciocínio para a resolução de problemas.

De uma forma geral, os resultados produzidos foram bastante satisfatórios e enriquecedores para todos os elementos do grupo pois permitiu consolidar conhecimentos adquiridos na unidade curricular.