Trabalho Prático Nº2 – Gateway Aplicacional e Balanceador de Carga sofisticado para HTTP

Duração: 6 aulas (não consecutivas, ver planeamento das aulas práticas no e-learning)

Motivação

São várias as razões que podem levar instalação de um *gateway* aplicacional que por definição interliga dois protocolos de aplicação distintos. Desde a segurança (ex: controlo de acesso, isolamento da rede interna) ao desempenho (ex: reduzir interações, balancear carga), passando pela flexibilidade de poder gerir o serviço em função de parâmetros que só estão disponíveis na camada de aplicação, como por exemplo o tipo de cliente, o tipo de pedido, o formato de dados requerido, a língua, etc. Não se pode confundir um *gateway* e aplicação com um balanceador de carga ou um reverse proxy, que tipicamente operam na camada 4 (transporte). O objetivo de um balanceador de carga é sobretudo melhorar a escalabilidade e disponibilidade de um serviço, permitindo a distribuição de pedidos de um servidor para muitos outros, normalmente de forma transparente, sem quaisquer ingerências na camada aplicacional. O mesmo se pode dizer de um reverse-proxy, que pretende adicionalmente melhorar a segurança e o desempenho, recorrendo por exemplo a *caching*. Ao operar na camada 7, um *gateway* de aplicação desencapsula dados de um protocolo de aplicação e encapsula noutro, ganhando com isso a flexibilidade de poder fazer as coisas de maneira diferente.

Objetivos

O objetivo deste trabalho é implementar um *gateway* de aplicação, designado por **HttpGw,** que opere exclusivamente com o protocolo HTTP/1.1 e que seja capaz de responder a múltiplos pedidos em simultâneo, recorrendo a a uma pool dinâmica de N servidores de alto desempenho, designados por **FastFileSrv**, e usando um protocolo a especificar para o efeito, conforme ilustrado na Figura 1. Por se tratar de um exercício essencialmente pedagógico, assumem-se à priori algumas simplificações: os dados a disponibilizar são ficheiros (texto, imagens, áudio, vídeo) de vários tamanhos, que são solicitados apenas por pedidos HTTP GET simples. O *gateway* de aplicação a desenvolver, recebe pedidos HTTP dirigidos à porta 80, e toma consciência do ficheiro que é pedido por *parsing* da primeira linha do *HTTP Request*. Provavelmente basta a primeira linha, mas nada o impede de processar todo o cabeçalho do pedido. Depois solicita a um ou mais dos servidores disponíveis na *pool* de servidores **FastFileSrv** que conhece, informação de cabeçalho (metadados) e blocos de dados (aqui designados por *chunks*). A forma como distribui a carga está em aberto. Mencionam-se apenas dois extremos possíveis: i) escolher um servidor e pedir tudo ao mesmo servidor sequencialmente; ou ii) escolher todos os servidores disponíveis e pedir em separado um *chunk* a cada um, ou o mesmo chunk a mais que um servidor, para tolerância a faltas. A comunicação entre o **HttpGw** e os servidores de suporte **FastFileSrv** deve ser feita de acordo com um protocolo especificamente desenhado para o efeito, a funcionar sobre UDP e não orientado à conexão e sem informação de estado. Os servidores **FastFileSrv** servem todos os mesmos ficheiros, e podem ser parados ou arrancados a qualquer instante, se necessário, sem que isso tenha impacto no serviço prestado pelo **HttpGw**. O **HttpGw** pode recusar pedidos dos clientes por questões de segurança (IP de origem ou excesso de pedidos por segundo).

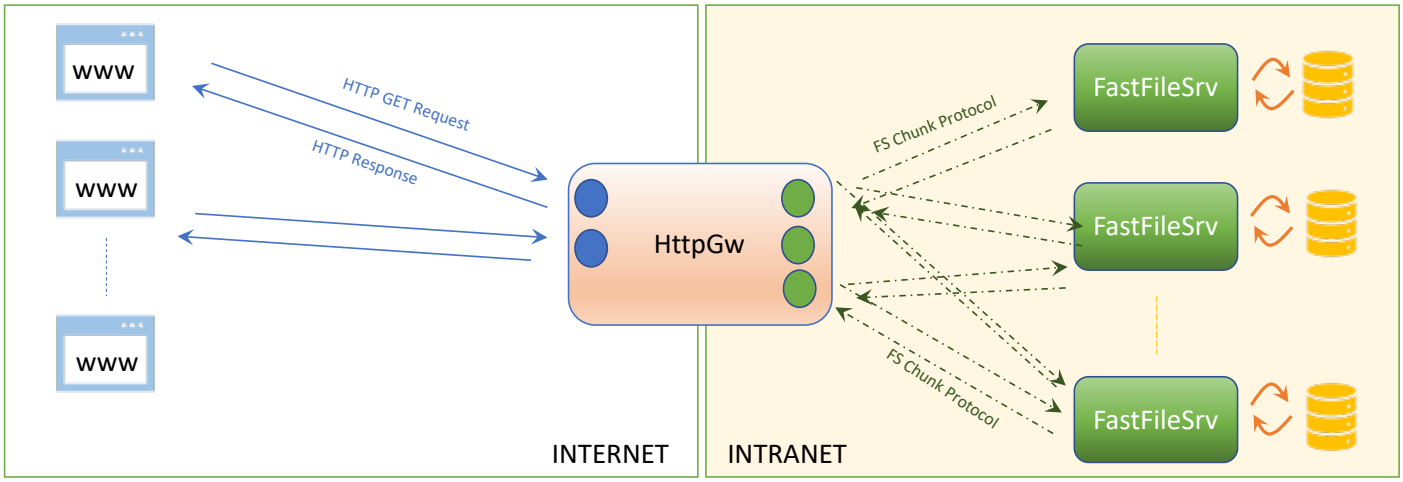

Figura 1: Esquema geral de funcionamento

Descrição

Para facilitar o desenho do sistema, explicita-se na Figura 2 um possível cenário de teste, que a seguir se detalha. O primeiro passo é instalar e executar o **HttpGw** num *host*. O **HttpGw** deve escutar novos pedidos *HTTP GET Request* na porta de atendimento 80 sobre TCP, e, em simultâneo, escutar na porta 80 em UDP[1] por PDUs (*Protocol Data Units*) de um novo protocolo designado por *FS Chunk Protocol* , a desenhar especificamente para o efeito. Na porta 80 TCP o HttpGw recebe pedidos HTTP. Por cada conexão aceite, pode criar uma *thread* ou um processo

[1] A porta 80, quer em TCP quer em UDP, é uma porta reservada, para a qual pode ser necessário ter privilégios de super user (root). As portas sugeridas podem ser trocadas por outras para maior comodidade e facilidade de testes (ex: TCP 8080 e UDP 8888)

trabalhador que se encarregue de: i) fazer o parsing do pedido HTTP GET para extração do nome do ficheiro solicitado; ii) pedir os metadados desse ficheiro a um ou mais dos servidores **FastFileSrv**; iii) executar um algoritmo de descarga do ficheiro de modo a pedir todos os *chunks* necessários a um ou mais servidores **FastFileSrv** que estejam ativos (*FS Chunk Protocol*); iv) reunir todos os *chunks* e devolver a resposta HTTP ao cliente; v) terminar a conexão.

De igual modo, para que o serviço possa funcionar como um todo, é necessário instalar um ou mais servidores **FastFileSrv** em várias máquinas de suporte. Cada um destes servidores deve ser executado indicando na linha de comando o endereço e a porta do servidor **HttpGw** a quem vão disponibilizar o serviço (Exemplo: **$> FastFileSrv 10.1.1.1 80**). Podem ser indicados outros parâmetros de configuração na linha de comando. A primeira coisa que o servidor **FastFileSrv** deve fazer é ficar à escuta de pacotes *FS Chunk Protocol* numa porta de serviço, que por omissão é também a 80 em UDP. De seguida, regista-se junto do **HttpGw**, indicando o seu IP e porta, para que este o possa incluir numa lista de servidores ativos. Neste registo pode ser útil implementar alguma forma de autenticação simples, de modo a evitar falsos servidores **FastFileSrv** e/ou falsos servidores **HttpGw**. Este requisito é opcional.

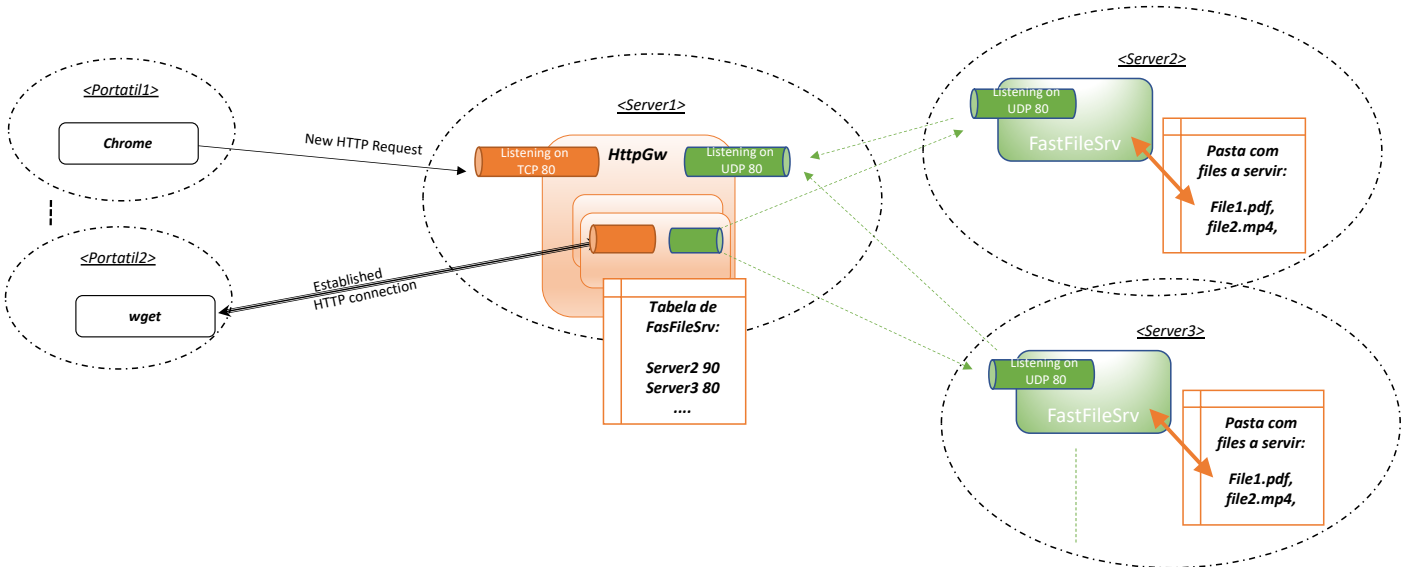

Figura 2: Visão detalhada

Pode-se usar como plataforma de teste o emulador CORE e a topologia CC-Topo-2021.imn (embora tal não seja obrigatório). Para o cenário de teste mínimo é necessário arrancar o servidor **HttpGw** e dois ou mais servidores **FastFileSrv** em diferentes localizações:

Server1> HttpGw
Ativo em 10.1.1.1 porta 80

Server2> FastFileSrv 10.1.1.1 80
Server3> FastFileSrv 10.1.1.1 80
Marte> FastFileSrv 10.1.1.1 80
....
Portatil1> wget http://10.1.1.1:80/um-ficheiro-grande.pdf & wget http://10.1.1.1:80/outra-file-ao-mesmo-tempo.mp4

Todos os servidores **FastFileSrv** devem ter acesso aos ficheiros usados no teste (um-ficheiro-grande.pdf e outra-file-ao-mesmo-tempo.mp4)

Desenvolvimento

Requisitos obrigatórios:

- O sistema deve conseguir responder a pedidos HTTP GET de forma fiável e robusta
- O sistema deve conseguir atender múltiplos pedidos em simultâneo
- O sistema deve ser eficiente
- O sistema deve permitir entrada e saída de novos servidores FastFileSrv a qualquer momento sem interrupção de serviço

Requisitos opcionais

- Segurança (autenticação e controlo de acessos)
- Resposta a pedidos HTTP 1.1 persistentes e/ou HTTPS

Sugere-se que o desenvolvimento tenha duas fases distintas, uma mais focada no novo protocolo a propor sobre UDP e outra na concretização do gateway aplicacional (mas é apenas uma sugestão).

FASE 1: Desenho e teste do protocolo FS Chunk

Etapas sugeridas para esta fase:

- Especificar o protocolo *FS Chunk Protocol* para funcionar sobre UDP
  - formato das mensagens protocolares (sintaxe)
  - função e significado dos campos (semântica)
  - diagrama temporal ilustrativo (comportamento)
- Implementação e teste do protocolo *FS Chunk Protocol*
  - exemplo de teste: pedir, estaticamente, um conteúdo pequeno

FASE 2: Implementação do gateway de aplicação HttpGw

Etapas sugeridas para esta fase:

- Aceitar pedidos HTTP GET na porta TCP 80
  - Parsing do pedido
  - Pedir metadados e os chunks
  - Devolver resposta HTTP Response
- Teste no cenário proposto

Relatório

O relatório deve ser escrito em formato de artigo com um máximo de 10 páginas (recomenda-se o uso do formato LNCS - *Lecture Notes in Computer Science*, instruções para autores em http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0). Deve descrever o essencial do desenho e implementação com a seguinte estrutura recomendada:

- Introdução
- Arquitetura da solução
- Especificação do protocolo
  * formato das mensagens protocolares
  * interações
- Implementação
  * detalhes, parâmetros, bibliotecas de funções, etc.
- Testes e resultados
- Conclusões e trabalho futuro

Regras de submissão

Cada grupo deve fazer a submissão (*upload*) do trabalho, usando a opção de "troca de ficheiros" associada ao seu grupo nos "Grupos" do Blackboard (*elearning.uminho.pt*), da seguinte forma:

- **CC-TP2-PLx-Gy-Rel.pdf** para o relatório
- **CC-TP2-PLx-Gy-Codigo.zip** ou **CC-TP2-PLx-Gy-Codigo.rar**) para a pasta com o código desenvolvido (devidamente documentado)

em que **x** é o identificador do turno PL e **y** o número do grupo (e.g. CC-TP2-PL2-G2-Rel.pdf para o relatório TP2 do grupo 2 do PL2)