



Universidade do Minho

Comunicações por Computador

MIEI - 3º ANO - 2º SEMESTRE

UNIVERSIDADE DO MINHO

GATEWAY APLICACIONAL E BALANCEADOR DE CARGA SOFISTICADO PARA HTTP

PL3 - Grupo 7



Renato Cruzinha
A75310

Braga, 25 de maio de 2021

Conteúdo

1	Introdução	2
2	Arquitetura da Solução	2
2.1	HttpGateway	3
2.2	FastFileSrv	3
2.3	FSChunk	3
3	Especificação do Protocolo	4
3.1	FSChunk Protocol	4
3.2	Fluxo da Troca de Mensagens	4
4	Implementação	5
4.1	Thread UDP	5
4.1.1	FastFileServeTable	6
4.2	Thread TCP	7
4.3	Tratamento de Pedidos	7
4.4	Aceitar Multiplos FastFileSrv	8
5	Testes e Resultados	8
6	Conclusões	10

1. Introdução

Neste trabalho foi nos proposto que desenhássemos e implementássemos um gateway aplicativo e balanceador de carga sofisticado para Http. Para isso foi necessário criar um protocolo exclusivo do nosso gateway, para comunicar internamente com os seus servidores.

Para este trabalho é importante ter em conta que estamos a falar ao nível da camada de aplicação, pelo que o protocolo é definido por nós dentro da aplicação.

O nosso gateway tem duas Threads a correr com as sockets abertas, de TCP na porta 8080, e de UDP na porta 8888.

Para a realização do mesmo, foi utilizada a linguagem *Python* para programar todo o código necessário para a produção do gateway aplicativo e também utilizada a ferramenta do *CORE* para testar o bom funcionamento do mesmo.

2. Arquitetura da Solução

De modo a desenvolver o nosso Gateway, tivemos à partida de criar um modelo arquitetónico onde estão desenhados a estrutura das funcionalidades requeridas.

Para isso, foi utilizada a arquitetura proposta no enunciado do trabalho prático:

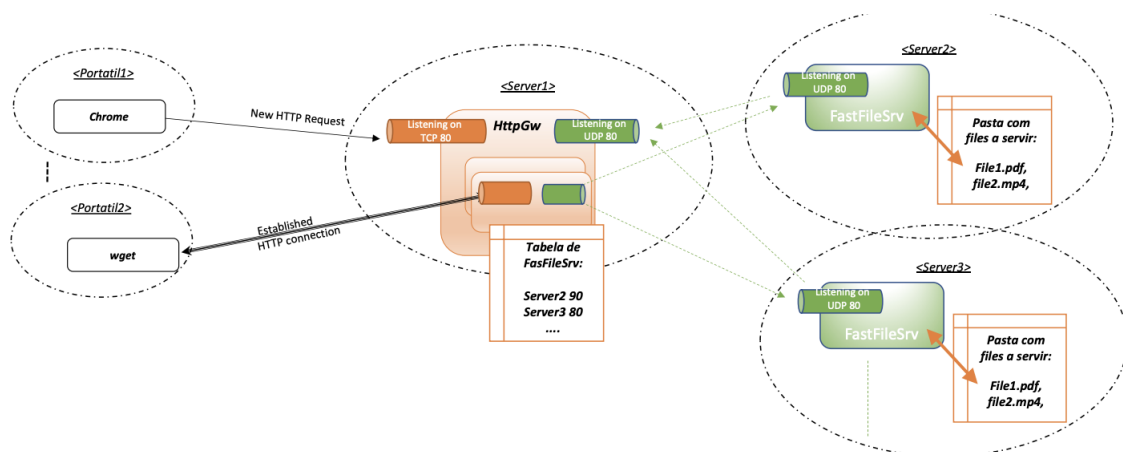


Figura 2.1: Visão detalhada

Figura 2.1: Arquitetura do HttpGateway

2.1 HttpGateway

Este é o servidor que atende a todos os pedidos, quer vindos do TCP, quer do UDP, e como tal, é necessário correr em simultâneo vários processos.

São então criadas duas Threads no momento do início do nosso balanceador de carga. Uma fica a correr os processos de TCP, vindos através de um HTTP request, em que é parsado o pedido e a Thread fica em espera até o ficheiro ser entregue. A segunda Thread é a que corre a ligação UDP, que estabelece uma ligação a cada FastFileServ que se ligue ao nosso gateway. Assim sendo, aquando do início de um FastFileServ, este envia uma mensagem protocolar, através de FSChunk (protocolo definido para o propósito), em que envia todos os ficheiros contidos na maquina que se acabou de ligar, para poder ser adicionada essa informação à base de conhecimento do gateway.

Assim, o HttpGateway corre duas Threads, em que estão sinalizadas e sincronizadas para correrem de modo a que o ficheiro que esteja a vir de um FastFileServ possa prosseguir para o HTTP response sem nenhuma falha de informação.

Posteriormente neste relatório será apresentado todo o workflow do gateway.

2.2 FastFileSrv

De modo ao gateway ter máquinas a ligarem-se para poder responder com os ficheiros, foi preciso criar os FastFileServe (FFS). Tal como o nome pode indicar, são servidores que possuem nas suas máquinas ficheiros que sejam pretendidos pelos clientes, e se ligam ao gateway para o poderem servir com os ficheiros desejados. Todas as comunicação que os FFS fazem são exclusivamente com o gateway, e todas elas são feitas via UDP. Para além disso, todos os datagramas são enviados através do protocolo FSChunk, tal como acontece do outro lado. Assim, é estabelecida uma ligação no arranque do servidor e o ffs fica à espera de receber um nome de um ficheiro para poder mandar para o gateway. Todo o processo de divisão de chunks é feito dentro do próprio ffs, que fica a enviar mensagens até o ficheiro estar completo. Do outro lado, o gateway já está preparado para este processo, tendo conhecimento do número de mensagens que vai receber e portanto preparado para intervir caso algo esteja a correr fora do normal.

2.3 FSChunk

Aqui é onde é definido o protocolo a usar nos datagramas UDP. Este protocolo basicamente é um chunk dos ffs e é o responsável pela comunicação aplicacional entre o HttpGateway e os FastFileSrv. Em todas as mensagens trocadas nessa conexão estão presentes as informações da sua origem e destino, com os ip e portas do httpgw e ffs correspondentes, assim como um limitador para o máximo de tamanho do chunk e por fim a mensagem em si a ser enviada.

3. Especificação do Protocolo

3.1 FSChunk Protocol

Tal como referido acima, foi criado um protocolo de troca de mensagens na conexão feita via UDP. Para isso, foi criada uma nova classe que continha as seguintes informações:

```
class FSChunk:
    def __init__(self, ffs_IP, httpgw_IP, data):
        self.max_chunksize = constants.MAX_CHUNKSIZE      # bytes
        self.ffs_IP = ffs_IP                             # (IP, Porta) do FastFileServ
        self.httpgw_IP = httpgw_IP                       # (IP, Porta) do HttpGw
        self.data = data                                  # Mensagem
```

Figura 3.1: FSChunk Protocol

Como podemos ver na imagem, são guardadas os tuplos (IP, Porta) quer do FFS, quer do HttpGateway, assim como a mensagem que será enviada na ligação da socket. Isto permite saber sempre qual o ffs que está a receber a informação para depois poder enviar novamente uma resposta de volta, caso seja necessário.

3.2 Fluxo da Troca de Mensagens

Para efetuar a troca de mensagens, foi preciso dar encode de objetos e passá-los como bytes através da ligação UDP, que permite transmitir até 1024b. Para isso, foi utilizada uma biblioteca externa *pickle*, que permitiu enviar com bastante mais facilidade fazendo uso do FSChunk Protocol. Sempre que fosse enviada uma mensagem, era criada uma nova instância de FSChunk para poder enviar as mensagens através do protocolo criado.

```
fchunk = FSChunk.FSChunk(IP_ADDRESS, IP_ADDRESS, str.encode(msg))
UDPClientSocket.sendto(pickle.dumps(fchunk), (IP_ADDRESS, UDP_PORT_NO))
```

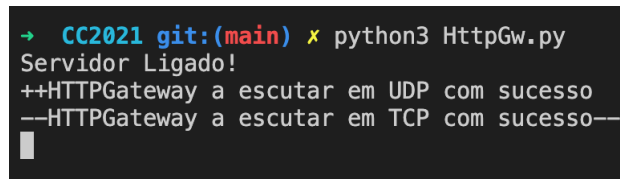
Figura 3.2: Criação de um FSChunk

4. Implementação

De modo a iniciar o nosso gateway aplicacional e balanceador de carga sofisticado para http, deve correr o comando python que permite correr a main do programa.

```
python3 HttpGw.py
```

Assim, mal seja iniciado o servidor, são postas logo duas Threads a correr para cada uma das ligações que queríamos que existissem, ou seja, uma TCP, e uma UDP.



```
→ CC2021 git:(main) x python3 HttpGw.py
Servidor Ligado!
++HTTPGateway a escutar em UDP com sucesso
--HTTPGateway a escutar em TCP com sucesso--
```

Figura 4.1: HttpGateway Iniciado

Nota: Todos os prints usados no terminal do gateway, para identificar de qual das conexões é feito o debug, sempre que for um print de algo vindo do FFS inicia o print com ++ e caso seja um print da thread do TCP é iniciado com - -

4.1 Thread UDP

Logo que é iniciada a Thread é feita a ligação da conexão, ou seja, o gateway abre a porta 8888 para escutar tudo o que venha em UDP.



```
# Faz a ligação UDP
def UDPListen(lock):
    IP_ADDRESS = "127.0.0.1"
    UDP_PORT_NO = 8888

    UDPServerSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # IPv4, UDP
    UDPServerSocket.bind((IP_ADDRESS, UDP_PORT_NO))
```

Figura 4.2: Iniciar Socket UDP

A primeira mensagem a ser enviada é assim que um FFS se ligue ao gateway. Esta mensagem contém uma lista com todos os nomes de ficheiros que a máquina tem na pasta com permissão para aceder, e também à frente tem os tamanhos de cada ficheiro, porque durante a implementação senti a necessidade de ter essa informação já do lado do gateway. Para além

disso, foi definido também um *max_chunksize* constante de 500b, para saber quanta informação seria possível enviar em cada datagrama. Para o caso de a máquina ter muitos ficheiros e não conseguir enviar o tuplo (filename, filesize) de todos, vai precisar de enviar de seguida outra mensagem com a informação dos restantes ficheiros. Sendo assim, foi necessário calcular primeiro quantas mensagens iriam ser enviadas para o gateway assim que o ffs se ligue, e esse número é enviado na primeira mensagem e decrementado a cada mensagem enviada, de modo ao gateway saber quando vai receber a última mensagem com nomes de ficheiros, para depois poder guardar essa informação toda e juntá-la.

```
→ CC2021 git:(main) x python3 FastFileSrv.py
Meus Ficheiros: [('FastFileServeTable.py', 1635), ('.DS_
Store', 6148), ('HttpGw.py', 5813), ('teste.txt', 169),
('constants.py', 53), ('FastFileSrv.py', 2421), ('README
.md', 46), ('FSChunk.py', 502)] 8
Enviei os meus ficheiros ao HttpGw
```

Figura 4.3: Primeira Mensagem FFS

Do lado do gateway, assim que chegue a mensagem é preciso recebê-la, parse-la e guardar os dados em estruturas de dados. Para tal, serão mostrados em baixo o modo como são guardados os dados em objetos.

4.1.1 FastFileServeTable

Para guardar os dados de modo a que o gateway os possa processar mais tarde, foi criada uma FastFileServeTable, em que guarda um dicionário de servidores conetados ao gateway, guardando o par (ip,[files]), em que associa a cada ip uma lista de ficheiros. De modo a encapsular e facilitar o acesso a estes dados, estes não foram guardados diretamente como (ip,[files]) mas sim como uma entrada na tabela, ou seja, um FastFileServeTableEntry, que associa um ip a uma lista de ficheiros.

```
class FastFileServeTable:
    def __init__(self):
        self.servidores = {}
```

Figura 4.4: Criação de uma FastFileServeTable

```
class FastFileServeTableEntry:
    def __init__(self, ip, porta, files):
        self.ip = ip
        self.porta = porta
        self.files = files
```

Figura 4.5: Criação de uma FastFileServeTableEntry

4.2 Thread TCP

Assim que seja iniciado o nosso gateway aplicacional, a par do que acontece no UDP, também o TCP estabelece logo uma socket para ficar a ouvir na porta 8080.

```
# Faz a ligação TCP
def TCPListen(lock):

    IP_ADDRESS = "localhost"
    TCP_PORT_NO = 8080

    TCPServerSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # IPv4, TCP
    TCPServerSocket.bind((IP_ADDRESS, TCP_PORT_NO))
```

Figura 4.6: Iniciar Socket TCP

Após estar estabelecida a conexão TCP com o balanceador de carga, este fica a escutar na porta até cair algum pedido HTTP GET request. Assim que um pedido http seja enviado ao gateway, este parse os headers do http request e apanha o nome do ficheiro que vai ter que pedir aos ffs para devolver num HTTP response.

A seguir a ter acontecido o pedido, o gateway adiciona o pedido a mais um estrutura de dados usada para guardar todos os pedidos. Sendo assim, o gateway associa a um (ip, porta) um nome de ficheiro que este esteja à procura.

A partir daí, a Thread TCP fica à espera que o UDP vá fazer o pedido do ficheiro aos ffs, pois caso avance, depois a variável global utilizada para guardar todo o ficheiro a mandar vai ser lida numa ordem cronológica da que vai ser escrita pela thread do UDP, pelo que tem de existir sincronização entre as Threads.

4.3 Tratamento de Pedidos

Assim que seja guardado o pedido na lista de pedidos guardada no gateway, é processado a partir do primeiro pedido. Assim, é lido o nome do ficheiro e de seguida é feita uma função de procura para saber quais são os ffs disponíveis para pedir o respetivo ficheiro. Após isto, é selecionado um ffs para pedir o ficheiro. Este vai ser o ffs responsável pelo ficheiro inteiro, ele é responsável por dividir em chunks pelo tamanho de ficheiro e saber quantas mensagens vai ter de enviar para o gateway. Aqui, foi onde foi necessário ter guardado o tamanho do ficheiro no lado do gateway, porque assim foi possível calcular logo quantas mensagens vão ser enviadas pelo ffs, de modo a no fim podermos também fazer uma verificação, através do tamanho em bytes da mensagem formulada por todos os chunks enviados pelo ffs e também através do número de mensagens enviadas pelo mesmo. Deste modo, se algo correr fora do previsto, é possível limpar o estado da variável que guarda o ficheiro temporariamente e pedir novamente a um ffs diferente da lista de ffs disponíveis para aquele ficheiro.

Para finalizar o tratamento do pedido, o gateway após verificar que está tudo de acordo com o ficheiro, faz a remoção do pedido da lista de pedidos e sinaliza a thread do tcp a dizer que o ficheiro está pronto a ser enviado. Assim, a thread tcp vai então à variável global onde está armazenado temporariamente o ficheiro e envia-o em modo de http response para a socket do tcp, com a porta do respetivo pedido.

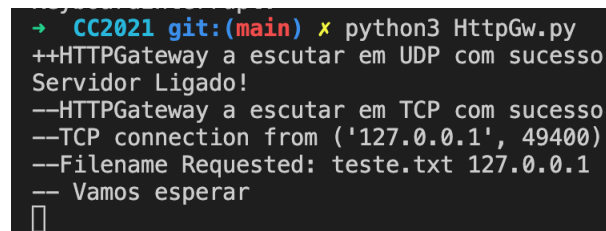
4.4 Aceitar Múltiplos FastFileSrv

Do modo que está configurado, o gateway pode aceitar múltiplos ffs ao mesmo tempo, pois sempre que um for ligado ao gateway, este é guardado com o (ip, porta) do ffs e a cada pedido é analisado todos os ffs que tenham entrado até então, pois adicionar e remover um ffs a qualquer momento do nosso projeto.

5. Testes e Resultados

Para testar o workflow da nossa aplicação, metemos o gateway a correr e em seguida fomos fazer um pedido http para ver se estava a adicionar à lista de pedidos.

Como podemos confirmar pela imagem abaixo, o gateway deixa o tcp em espera enquanto não existe nenhum ffs capaz de satisfazer o ficheiro pedido.



```
→ CC2021 git:(main) x python3 HttpGw.py
++HTTPGateway a escutar em UDP com sucesso
Servidor Ligado!
--HTTPGateway a escutar em TCP com sucesso
--TCP connection from ('127.0.0.1', 49400)
--Filename Requested: teste.txt 127.0.0.1
-- Vamos esperar
█
```

Figura 5.1: Introdução de um pedido

De seguida, foi introduzido um ffs no gateway, com o ficheiro em teste na sua diretoria, para garantir que a máquina continha o ficheiro pretendido para resolver o pedido.

Desta vez, conseguimos testar várias coisas ao mesmo tempo. Podemos confirmar que o ffs mal se liga ao httpgateway envia a primeira mensagem com os ficheiros todos que tem disponíveis. Podemos ver ainda que é adicionado com sucesso o ffs à FastFileSrvTable. Conseguimos comprovar ainda que o está a ir buscar o ficheiro de modo correto, pelo tamanho do mesmo. De seguida vemos que é enviado o ficheiro certo e que o gateway o recebe também certo em bytes, e que no fim remove o pedido para enviar o http response. Podemos ainda confirmar que o ffs fica à espera de receber um novo ficheiro, ou então pode terminar a sua sessão e fechar a conexão estabelecida com o httpgateway.

```

--TCP connection from ('127.0.0.1', 49400)
--Filename Requested: teste.txt 127.0.0.1
-- Vamos esperar
++Message b"1__[('FastFileServeTable.py', 1635), ('.DS_Store', 6148), ('HttpGw.py', 5811), ('teste.txt', 169), ('constants.py', 53), ('FastFileSrv.py', 2421), ('README.md', 46), ('FSChunk.py', 502)]"
++FastFileServ ('127.0.0.1', 61749) Ligado 1 [('FastFileServeTable.py', 1635), ('.DS_Store', 6148), ('HttpGw.py', 5811), ('teste.txt', 169), ('constants.py', 53), ('FastFileSrv.py', 2421), ('README.md', 46), ('FSChunk.py', 502)]
++Lista de servers {'127.0.0.1': <FastFileServeTable.FastFileServeTableEntry object at 0x7fe8c3919e80>}
++Lista Pedidos True
++ENTROU
{'127.0.0.1': <FastFileServeTable.FastFileServeTableEntry object at 0x7fe8c3919e80>}
++Disponiveis: [('127.0.0.1', <FastFileServeTable.FastFileServeTableEntry object at 0x7fe8c3919e80>)]
++Cenas Ficheiro: teste.txt ('127.0.0.1', <FastFileServeTable.FastFileServeTableEntry object at 0x7fe8c3919e80>)
169
++fora
++ Removido pedido b'Isto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste'
-- Parei de esperar

```

Figura 5.2: Teste HttpGw.py

```

→ CC2021 git:(main) x python3 FastFileSrv.py
Meus Ficheiros: [('FastFileServeTable.py', 1635), ('.DS_Store', 6148), ('HttpGw.py', 5811), ('teste.txt', 169), ('constants.py', 53), ('FastFileSrv.py', 2421), ('README.md', 46), ('FSChunk.py', 502)] 8
Enviei os meus ficheiros ao HttpGw
Message from Server Ficheiro: teste.txt
b'Isto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste\nIsto \xc3\xa9 um teste'
addr: ('127.0.0.1', 8888)
sent filechunk
Pressione 'Ctrl+C' para desligar o FFS ou aguarde novo pedido de ficheiro
^C
A desligar FastFileServ...

```

Figura 5.3: Teste FastFileSrv.py

6. Conclusões

Este trabalho foi deveras enriquecedor e repleto de desafios entusiasmantes com várias soluções possíveis e com vários problemas de engenharia que tivemos de fazer uso de todo o conhecimento adquirido ao longo não só desta UC, como também do curso. Sendo assim, podemos aprender mais sobre como aplicar transferências de pacotes ao nível da camada aplicacional, para além de ter tido a necessidade de aprender a linguagem *Python* e de ganhar conhecimento ao nível da ferramenta do *CORE*.

Como trabalho futuro, não posso deixar de realçar que gostaria de conseguir meter a thread do tcp a enviar o http response como prioridade a finalizar, pois seria o poder verificar na prática o httpgateway a funcionar com todos os requisitos mínimos. Uma melhor implementação surgiu-me após já ter muito deste projeto avançado e não consegui voltar atrás e refazer, mas gostaria de ter uma thread para cada ffs em vez de uma única a correr e cada ffs ter de esperar que o httpgateway processe os pedidos um por um. Estas seriam as minhas prioridades futuras a implementar neste projeto, contudo, num ponto mais avançado, poderia refazer os métodos de distribuição dos chunks de modo a fazer um melhor balanceamento ao invés de pedir um ficheiro a um único ffs.