Gateway Aplicacional e Balanceador de Carga sostificado para HTTP

Trabalho realizado por:

Bruno Filipe de Sousa Dias

Francisco Alves Andrade

Paulo Silva Sousa

A89513
Francisco Andrade

A89583
Bruno Dias

A89465
Paulo Sousa

Grupo 14
Comunicações por Computador 2020/2021
Universidade do Minho

Índice

1 Introdução

Este relatório é relativo ao trabalho prático nº2 da Unidade Curricular de *Comunicações por Computador*, onde foi desenvolvido um projeto alusivo ao Gateway Aplicacional e ao Balanceador de Carga sofisticado para HTTP.

O objetivo é a implementação de um *gateway* designado *HttpGw* que, operando exclusivamente com o protocolo HTTP/1.1, seja capaz de responder a múltiplos pedidos em simultâneo recorrendo a uma pool dinâmica de N servidores de alto desempenho, designados por *FastFileServer*.

Inicialmente focamo-nos no protocolo a propor sobre UDP, e depois passamos à concretização do *gateway* aplicacional.

Ao longo deste relatório iremos explicitar todo o desenvolvimento, tentando esclarecer todas as decisões tomadas ao longo do mesmo, até ao produto final.

2 Arquitetura da Solução

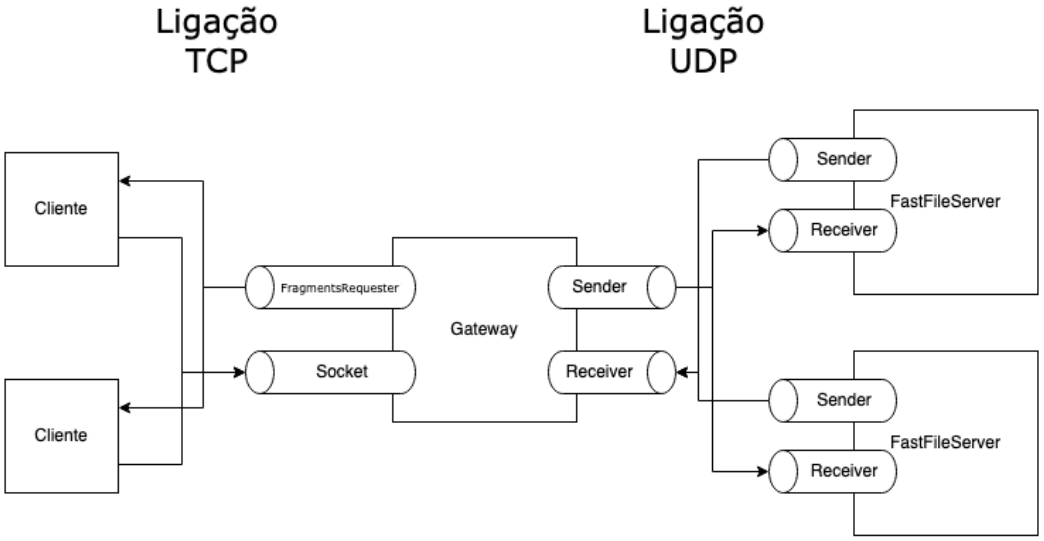

Figure 1: Arquitetura da Solução

3 Especificação do Protocolo

3.1 Formato das mensagens protocolares

No nosso pacote incluímos, para além do payload, o tipo de pacote de envio, o IP e Porta da origem e do destino e o ID do utilizador, caso este pacote tenha informação relativa a um Cliente.

Em seguida, temos uma tabela com toda a informação do pacote.

Informação no Pacote	
Tipo de Pacote	Identifica o tipo de informação que o pacote transporta e a função que este irá desempenhar
Porta de Origem	Identifica a Porta Origem de onde o Pacote foi emitido
Porta de Destino	Identifica a Porta Destino para onde o Pacote foi emitido
IP de Origem	Identifica o IP Origem de onde o Pacote foi emitido
IP de Destino	Identifica o IP Destino para onde o Pacote foi emitido
ID do User	Identificador do Cliente que fez o pedido (-1 se o pacote não estiver relacionado a um Cliente)
Chunk Transferência	Chunk a transferir
Payload	Dados

Table 1: Informação no Pacote

3.2 Tipos de Pacote

Na tabela 2, podemos encontrar os vários tipos de pacote existentes na nossa implementação:

Tipos de Pacote	
Tipo 1 (Verificação de Existência de Ficheiro)	Gateway pergunta ao FFs se o ficheiro existe.
Tipo 2 (Resposta afirmativa do FFs ao Gateway)	FFs responde ao Gateway que o ficheiro existe.
Tipo 3 (Resposta negativa do FFs ao Gateway)	FFs responde ao Gateway que o ficheiro não existe.
Tipo 4 (Pedido de ficheiro)	Gateway pede ao FFs um ficheiro que ele possui.
Tipo 5 (Envio de Ficheiro)	FFs envia ficheiro o requisitado ao Gateway.
Tipo 6 (Pedido de ligação do FFs)	FFs informa que se pretende ligar ao Gateway .
Tipo 7 (FFs desliga-se do Gateway)	FFs informa que se pretende desligar do Gateway.
Tipo 8 (Ligação bem sucedida)	Gateway informa que FFs se ligou corretamente.
Tipo 9 (Keep Alive)	GateWay envia mensagens esporadicamente para garantir que a ligação com o Servidor ainda se mantém ativa. Servidor responde o mesmo, caso isso se verifique.
Tipo 10 (Cancelamento de conexão)	Gateway confirma que a conexão foi cancelada.

Table 2: Tipos de Pacote

3.3 Interações

Nesta secção iremos descrever todo o tipo de interações realizadas no nosso sistema entre os clientes, o *Gateway* e os *FastFileServers*.

Começamos por apresentar uma transferência, como é possível visualizar através das figuras 2 e 3.

Em primeiro lugar, quando um utilizador efetua um pedido de transferência, o *Gateway* envia um pacote do tipo 1 com o path no payload.

O *FastFileServer* ao receber este pacote, verifica se o ficheiro existe na diretoria predefinida. Na eventualidade de o ficheiro não existir, é enviado um pacote do tipo 3 ao *Gateway* e este apresenta uma mensagem de erro ao utilizador. Caso contrário, o *FastFileServer* responde com um pacote do tipo 2, enviando no payload, para além do path, o tamanho do ficheiro em bytes. Neste caso, o *Gateway* calcula o número de chunks necessários para a fragmentação do pacote e pede ciclicamente a todos os *FastFileServers* existentes que lhe enviem uma porção do ficheiro através de um pacote do tipo 4, enviando o offset do ficheiro no payload junto com o path. Cada *FastFileServer* responde aos pacotes do tipo 4 que recebeu com um pacote do tipo 5, enviando no payload os bytes do ficheiro correspondentes à porção representada pelo offset recebido.

O *Gateway* aguarda que todos os chunks sejam enviados, expedindo posteriormente o ficheiro para o Cliente através de um *BufferedOutputStream*.

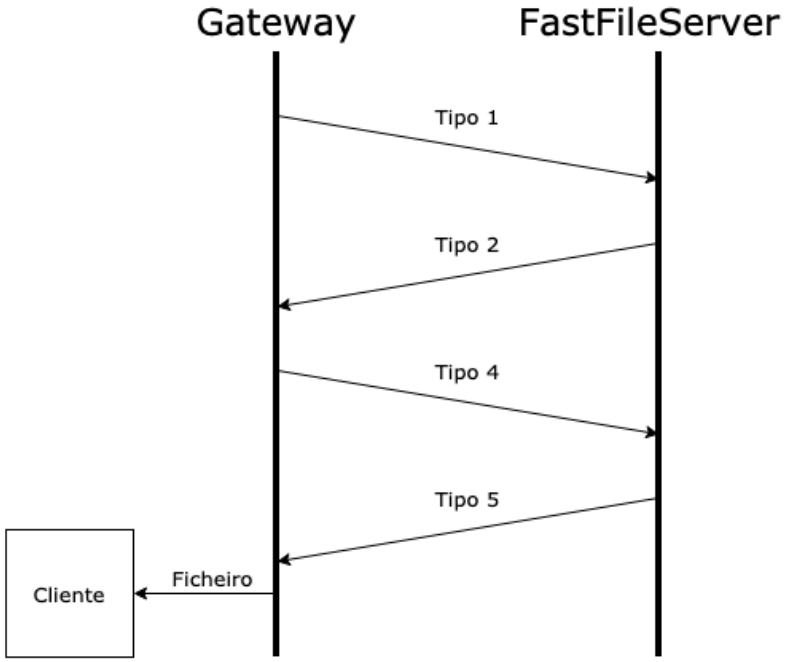

Figure 2: Transferência bem sucedida

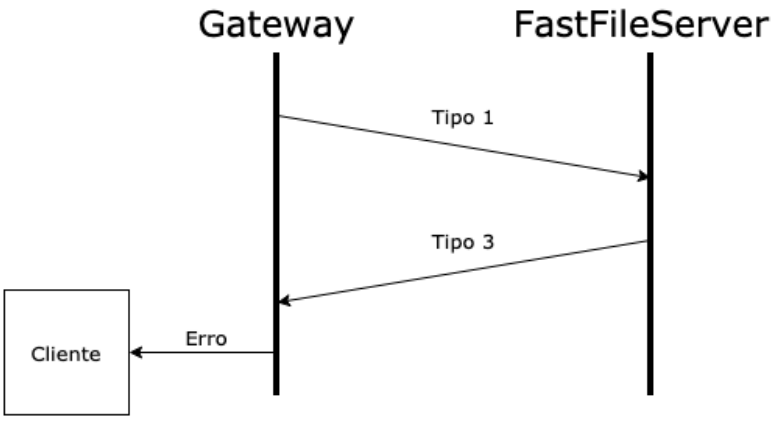

Figure 3: Transferência mal sucedida

Na figura 4 podemos ver o *FastFileServer* a estabelecer uma ligação com o *Gateway*. Quando um *FastFileServer* é iniciado, envia ao *Gateway* um pacote do tipo 6. O *Gateway* ao receber este pacote, adiciona o IP e a porta do servidor a um Map e envia de volta um pacote do Tipo 8 para garantir que a conecção foi bem estabelecida.

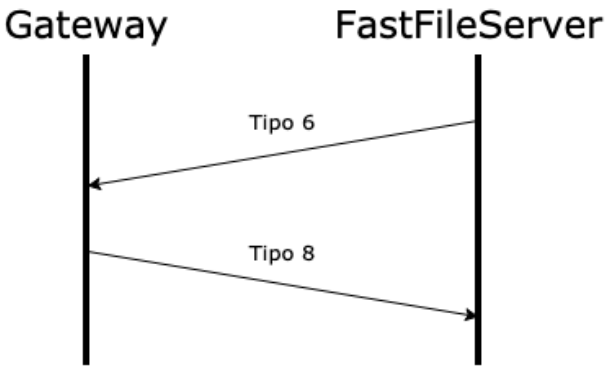

Figure 4: Estabelecimento de ligação entre o FFS e o Gateway

Na figura 5 podemos ver o *FastFileServer* a cancelar a ligação ao *Gateway*. Quando um *FastFileServer* é interrompido através da introdução da String *exit*, este envia ao *Gateway* um pacote do tipo 7. O *Gateway* ao receber este pacote, remove o servidor do Map de servidores e envia um pacote do tipo 10 a confirmar o encerramento da conecção. Quando o *FastFileServer* recebe este último pacote, interrompe o seu Receiver para não receber mais pacotes e fica em espera que o seu Sender acabe de esvaziar a queue de pacotes a enviar. Quando esta queue fica vazia, o *FastFileServer* desliga.

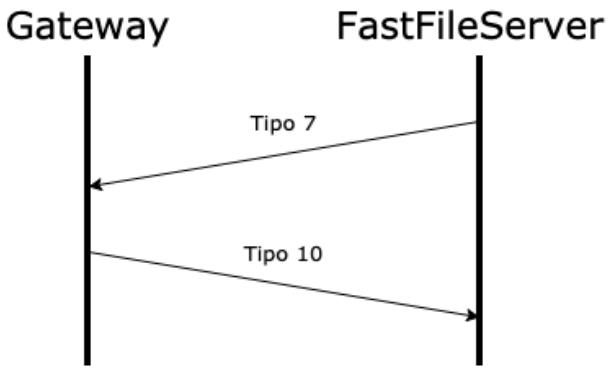

Figure 5: Encerramento de ligação entre o FFS e o Gateway

Na figura 6 podemos ver o processo de envio de beacons. Quando iniciamos o *FastFileServer*, criamos uma Thread responsável por enviar periódicamente beacons ao *Gateway*.

Quando o *Gateway* recebe um beacon atualiza a hora em que o servidor o enviou. Além disso, o *Gateway* tem uma Thread que verifica se o Servidor não envia um beacon à mais do que um certo tempo (está em idle) e, caso isto se verifique, remove-o do Map de servidores.

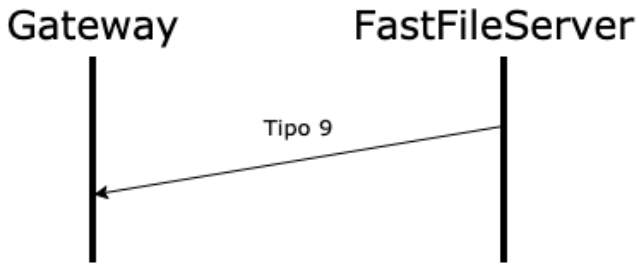

Figure 6:

4 Implementação

Nesta secção vamos demonstrar os aspetos mais significativos da nossa implementação da aplicação.

É de notar que a comunicação entre as Threads Sender e Receiver, tanto do *Gateway* como do *FastFileServer*, é assegurada por uma queue. Quando um Receiver recebe um *DatagramPacket* descompacta-o, processa a informação e cria um Packet de resposta para enviar. Este Packet é adicionado a uma queue e será posteriormente removido pelo Sender quando este estiver disponível para o enviar. Podemos também chamar a nossa queue de *priority queue* porque pacotes com maior importância, como é o exemplo dos beacons, são adicionados ao início da queue em vez de serem adicionados ao fim.

4.1 Gateway

Quando iniciamos o nosso *Gateway* são iniciadas três Threads: *BeaconGateway*, *ReciverGateway* e *SenderGateway*.

Após iniciar as Threads, o *Gateway* entra num ciclo em que aceita conecções TCP de clientes e processa os seus pedidos de transferência. Para isso, atribui um ID ao cliente (através de um contador), escolhe um server da lista de servidores disponíveis e cria um pacote do tipo 1 que adiciona à queue para ser enviado para o *FastFileServer*.

4.1.1 SenderGateway

A Thread *SenderGateway* é responsável por remover pacotes da queue (caso existam) e envia-lo para o respetivo *FastFileServer*. Para isso, após remover o pacote da queue, transforma-o num array de bytes, cria um novo DatagramPacket e envia-o para o *FastFileServer* através do DatagramSocket.

4.1.2 ReceiverGateway

A Thread *ReceiverGateway* é responsável por receber pacotes do *FastFileServer*, descompacta-los, processa-los e adicioná-los à queue para serem enviados.

Dentro dos diferentes pacotes que são processados nesta classe é de realçar o processamento de um pacote do tipo 2 (recebe informação de um ficheiro) e do tipo 5 (recebe chunk de um ficheiro).

Para o primeiro, recebemos como argumento um pacote anteriormente descrito com o tamanho do ficheiro em bytes no payload. Com este tamanho calculamos o número de chunks nessessário para transferir o ficheiro do *FastFileServer* e criamos um array com todos os inteiros de 1 ao número de chunks, do qual cada chunk vai ser removido à medida que chega no processamento do tipo 5.

Por último, executamos uma Thread chamada *FragmentsRequester* responsável por pedir os chunks ao servidor.

Para o segundo, o método recebe um packet com um chunk de um ficheiro, adiciona-o a um map de chunks onde todos os chunks são adicionados e remove o chunk do array descrito anteriormente.

FragmentsRequester

Nesta Thread, em primeiro lugar, é corrido um ciclo que verifica se todos os chunks já chegaram, verificando se o array com o número de cada chunk está vazio. Caso esteja, sai do ciclo, caso contrário é reenviado um pedido dos chunks em falta.

Assim que todos os chunks são recebidos, é aberto um *BufferedOutputStream* com o Socket do utilizador e é enviado ao Cliente o payload de todos os chunks. Após isso, o Socket é fechado para terminar a comunicação e o Cliente é removido da lista de utilizadores.

4.1.3 BeaconGateway

A Thread *BeaconGateway* é responsável por verificar que servidores é que se encontram em *idle* e removê-los da lista de servidores.

Para isso, percorre a lista de todos os servidores e verifica quais é que enviaram um Beacon pela última vez à mais de 15 segundos. Caso afirmativo, remove esse servidor da lista de servers.

4.2 FFServer

Quando iniciamos o nosso *FastFileServer* são iniciadas três Threads: *BeaconFFS*, *ReceiverFFS* e *SenderFFS*. Após a criação e o arranque das Threads, estas ficam responsáveis por fazer todo o trabalho do *FastFileServer*. Este fica, no entanto, a ler linhas do input até que seja introduzido como *input* a palavra "exit", de modo ao *FastFileServer* poder sair da forma correta e segura do sistema.

4.2.1 SenderFFS

A Thread *SenderFFS* fica, tal como a Thread *SenderGateway* é responsável por remover pacotes da queue (quando estes existem) e enviá-los para o *Gateway*. Este envio recorre a um DatagramSocket por onde é enviado um novo DatagramPacket que transporta os bytes de um pcaote removido da queue.

4.2.2 ReceiverFFS

A Thread *ReceiverFFS* fica responsável por receber os pacotes vindos do *Gateway*, passá-los por um processo de descompactação, processamento e terminando adicionando um Packet à queue, caso o *FastFileServer* necessite de eniar um Packet de resposta ao *Gateway*.

O *FastFileServer* pode receber pacotes de 4 tipos distintos.

O pacote 8 destina-se a confirmar que o *FastFileServer* se ligou corretamente ao Gateway. O pacote 10 confirma que a conexão com o *Gateway* foi bem encerrada e é crucial para o encerramento seguro desta Thread e do *FastFileServer*.

O pacote 1 pergunta ao *FastFileServer* se possui um certo ficheiro. O *FastFileServer* irá verificar a existência deste ficheiro. Caso o ficheiro exista, é colocado na Queue um Packet do tipo 2 que será enviado ao *Gateway* como forma de indicar que este ficheiro existe. No *payload* deste pacote, irá estar incluída a designação da diretoria e do ficheiro requisitado, bem como o tamanho do ficheiro em bytes, estando este separado por "#SIZE#". No caso de o ficheiro não existir, é adicionado à queue um Pacote do tipo 3 que indica ao *Gateway* que o ficheiro não existe, estando incluído no payload a designação da diretoria e do ficheiro requisitado.

O pacote do tipo 4 consiste na requisição de um pedaço do ficheiro. Assim, deste pacote será extraido qual o chunck do ficheiro que o *FastFileServer* deve enviar. Após isso, a Thread vai buscar os bytes de um certo offset e um certo tamanho calculados,

e coloca essa informação num novo Packet do tipo 5, onde inclui o chucnk que está a transferir, introduzindo os bytes desse chucnk no *payload* do pacote. Este pacote é então adicionado à queue, de modo a ser enviado ao *Gateway*.

4.2.3 BeaconFFS

A Thread *BeaconFFS* é responsável por enviar beacons ao *Gateway* de modo a garantir que o server não entrou em idle.

Para isso, a Thread *BeaconFFS* adiciona à queue, entre um definido intervalo de segundos, um Packet do tipo 9 que irá informar o *Gateway* que o *FastFileServer* do qual recebe este pacote ainda se encontra ativo e funcional. É imporatnte realçar que este Pacote é adicionado sempre no inicio da Queue para evitar casos de *starving* onde a Queue está demasiado cheia e o *Gateway* não recebe Beacons, considerando o FFS idle.

4.2.4 SafeExit

Quando é introduzido no terminal do *FastFileServer* o input *"exit"*, o servidor entra em processo de safe exit. Para isso, adiciona ao início da queue um pacote do tipo 7. Quando a resposta (na forma de um pacote do tipo 10) é recebida, as Threads *ReceiverFFS* e *BeaconFFS* são fechadas e o programa fica em *await* à espera o *SenderFFS* acabe de enviar todos os pacotes que tem na queue.

Uma vez que a queue esteja vazia, a Thread *SenderFFS* também termina e o servidor dá *exit(0)*.

4.3 Packet

Nesta classe, além dos aspetos referidos no tópico 2 deste relatório, é de realçar a implementação de encriptação dos Packets.

4.3.1 Encriptação

Por uma questão de segurança na transmissão de ficheiros, decidimos implementar encriptação nos nossos packets.

Para isso, criamos duas funções, uma para encriptar e outra para desencriptar. Nestas funções utilizamos as classes de java *Cipher* e *SecretKeySpec* para nos auxiliar na encriptação. Ambas as funções recebem e devolvem um array de bytes.

Após a criação das mesmas, no fim da execução da função *packetsToBytes* invocamos o método responsável por encriptar e no início do construtor Packet, com um array de bytes, invocamos o método responsável pela desencriptação. Assim, garantimos que os pacotes convertidos em arrays de bytes para transmissão contêm a mensagem encriptada.

5 Testes e Resultados

Neste capítulo iremos demonstrar o processo de execução do sistema e apresentar alguns resultados no carregamento dos ficheiros com diversos tamanhos.

Neste primeiro caso está representada a ligação dos *FastFileServers* ao *Gateway*. Além disso, também é possível verificar o envio de alguns *beacons* por parte dos *FastFileServers*.

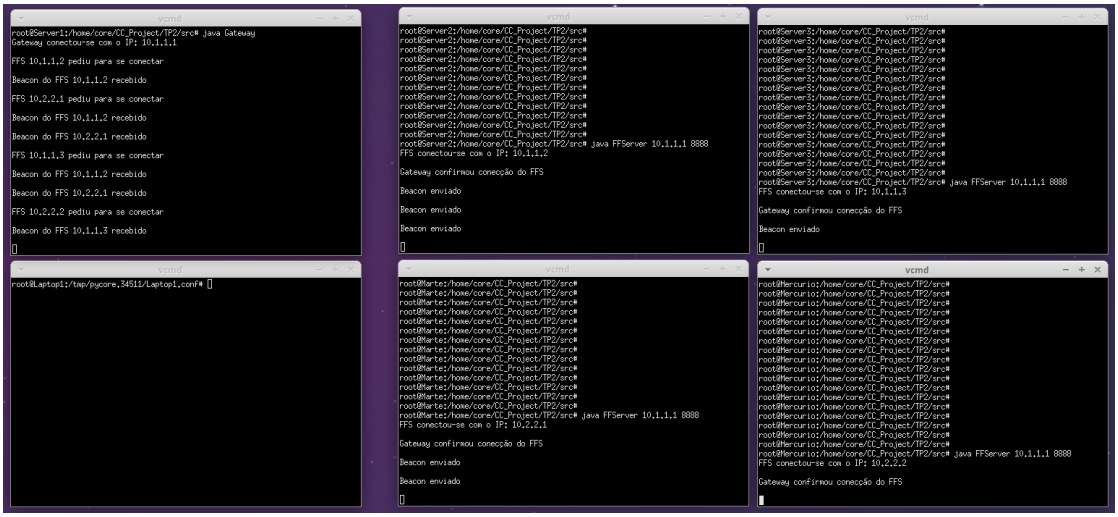

Figure 7: Ligação dos FastFileServeres ao Gateway

No caso seguinte está representado um pedido do ficheiro de menores dimensões *"verysmall.txt.1"* -> ficheiro de texto de 57 bytes apenas. Neste teste é possível ver a rotatividade de *FFServers*, uma vez que o pacote do tipo 1 é enviado a um servidor e o pacote do tipo 4 é enviado a outro.

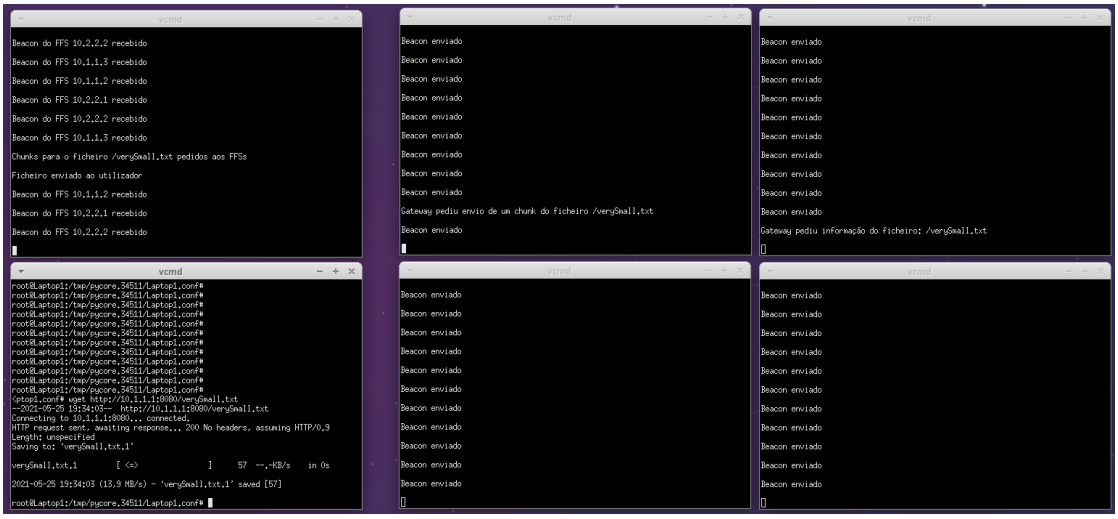

Figure 8: Pedido do Ficheiro de 57 Bytes

Neste caso apresentamos um pedido do ficheiro de maior dimensão *"videogrande.mp4"* -> ficheiro de vídeo (mp4) de 30 Megabytes.

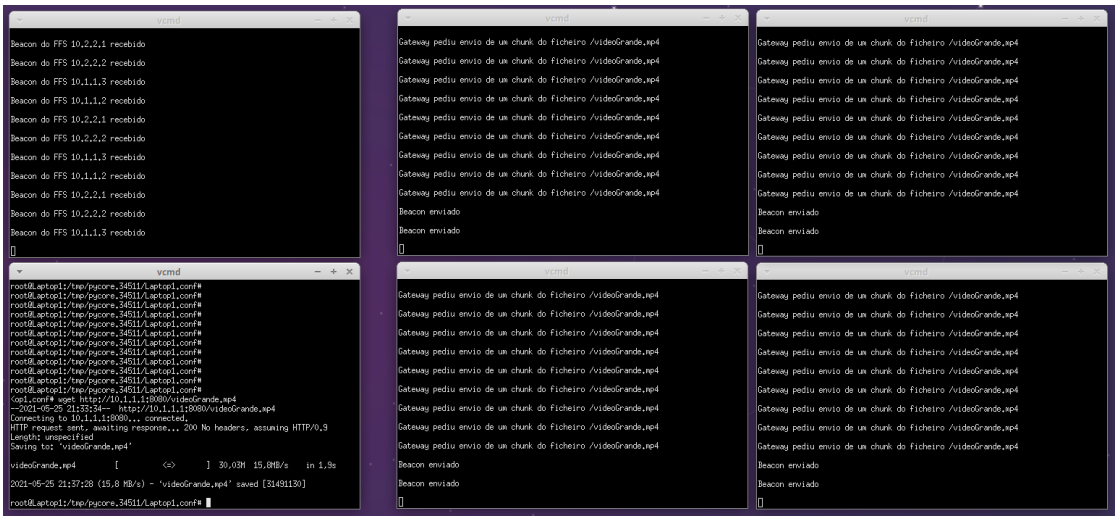

Figure 9: Pedido do Ficheiro de 30 Megabytes

Neste caso apresentamos um pedido de um ficheiro que não existe. Assim, o *Gateway* imprime uma mensagem de erro e o ficheiro descarregado para o Cliente apenas contém essa mensagem de erro.

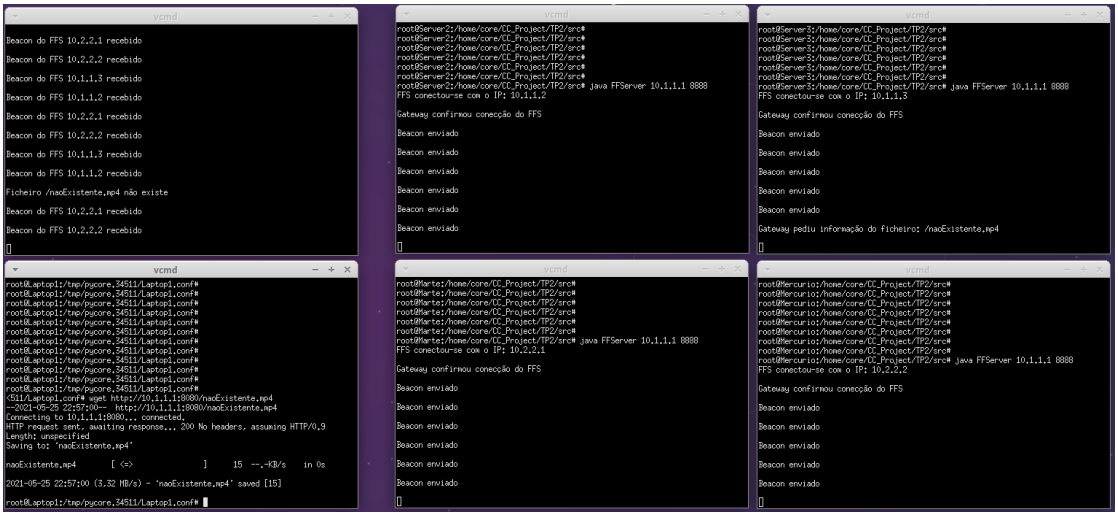

Figure 10: Pedido de Ficheiro Inexistente

Neste último caso, fazemos um pedido quando não existem servidores ligados. Deste modo, o *Gateway*, tal como no último caso, imprime uma mensagem de erro e o ficheiro descarregado para o Cliente contém a mensagem de erro.

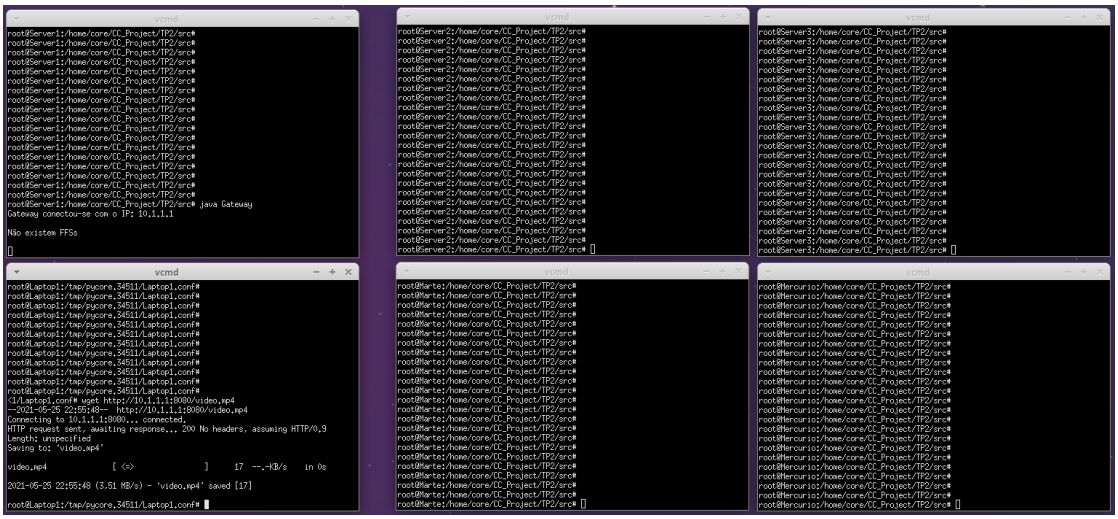

Figure 11: Pedido de Ficheiro sem sevidores ligados

6 Conclusões e Trabalho Futuro

Este trabalho prático revelou-se um grande desafio no sentido de termos de enfrentar um "Universo" completamente novo para nós e que nos deixou um pouco desorientados inicialmente. No entanto, após finalizado sentimo-nos bastante mais confiantes e seguros neste módulo.

Durante a implementação deparamo-nos com vários problemas e dificuldades que, com mais ou menos dificuldade, fomos resolvendo e tomando decisões na orientação do estado final do projeto.

Assim sendo, o trabalho prático revelou-se de extrema importância na assimilação e aperfeiçoamento dos conceitos lecionados ao longo das aulas e, ainda, no desenvolvimento de conceitos extra-aula que fomos adquirindo com a pesquisa extensa durante o desenvolvimento. Permitiu-nos, também, um aperfeiçoamento nas *skills* da linguagem de programação *Java* e na utilização de um emulador *core*, ferramenta esta que será útil no futuro com certeza.

Em suma, tendo em conta todo o trabalho realizado, estamos satisfeitos e orgulhosos com o produto final, e consideramos ir de encontro aquilo que foi solicitado.