



Universidade do Minho

Mestrado Integrado de Engenharia Informática

Comunicações por Computador

(2º Semestre - 2020/21)

Relatório do **TP2**

A89575 Joel Salgueiro Martins

A93785 Ricardo Miguel Santos Gomes

A89542 Carlos Filipe Coelho Ferreira

Braga

25 de maio de 2021

1. Introdução

Este relatório é relativo à elaboração do Trabalho Prático 2 (TP2) proposto na Unidade Curricular Comunicações por Computador com o tema *Gateway Aplicaçional e Balanceador de Carga sofisticado para HTTP*.

O principal objetivo deste trabalho é desenhar e implementar um serviço simples para servir ficheiros a clientes que se conectam ao *Gateway* via *TCP*, tendo o *Gateway* um número indefinido de *Fast File Servers* a ele ligado, com ficheiros prontos a ser servidos via *UDP*.

Portanto, numa primeira fase foi necessário desenhar e implementar um protocolo sobre *UDP* para criar e manter a interação do *Gateway* com os *Fast File Servers* e vice-versa. Depois, numa segunda fase, partiu-se para a implementação dum servidor chamado *Gateway*, que fica, no caso do nosso projeto, à escuta na porta 50000 (normalmente seria a porta 80, mas como esta é um porta do sistema operativo, resolvemos escolher outra, de modo a evitar problemas futuros), e que tratará de qualquer pedido *HTTP* recebido.

Desta forma, ao longo deste relatório iremos expor e explicar todo o processo que tivemos de percorrer até chegarmos ao resultado final, tirando partido da linguagem computacional *Java* para implementar todas as funcionalidades requeridas.

Palavras-Chave: Gateway, Fast File Servers (FFS), Java, TCP, UDP, HTTP, Comunicação de Computadores, Thread, ReentrantLock, ping.

2. Arquitetura da solução implementada

Para o desenvolvimento do problema apresentado foi necessário logo à partida estabelecer a arquitetura do nosso *Gateway* e *Fast File Server*, de forma a satisfazer todas as funcionalidades requisitadas. Desde modo, segue-se a arquitetura final implementada:

2.1. Gateway

Gateway será então a componente que tratará de servir diversos clientes e interagir com os diversos *Fast File Servers* nele registados.

- **Classe Gateway**

Esta classe é sem dúvida a componente central do *Gateway*, nela é definida a porta fixa a qual deve ser usada pelos clientes, assim como a porta que os *Fast File Server* devem usar para se puderem registar.

Antes de entrar no ciclo infinito à espera de pedidos dos clientes, esta classe irá criar uma *thread* *GatewayWorkerRegister* que irá tratar de toda a interação de controlo com os diversos *Fast File Server*. Depois de entrar no ciclo infinito, a cada conexão que um cliente estabeleça com o *Gateway* é criada também uma *thread* *HttpWorker* para tratar do seu pedido.

- **Classe GatewayWorkerRegister**

Esta classe trata, como já dito anteriormente, de toda a interação de controlo entre o *Gateway* e o *Fast File Server*, portanto, processa os pedidos de registo assim como verifica se cada *Fast File Server* registado se mantém ativo. Para isso, a *thread* *GatewayWorkerRegister* antes de entrar no seu ciclo infinito à espera de pedidos de registo, prepara uma nova *thread* *Ping* que será explicada já de seguida.

Continuando com o fluxo da *thread* *GatewayWorkerRegister*, esta após entrar no seu ciclo infinito, ficará à escuta na porta especificada pelo *Gateway* como a porta usada para registos, e a cada *pacoteRegisto* que o *Gateway* receba irá então guardar a informação necessária relativa ao *Fast File Server*.

A *thread* *Ping* entra também num ciclo infinito que será executado de segundo em segundo, de modo a verificar se os *Fast File Servers* se mantêm ativos e, para isso, apenas tratará de enviar um *PacotePedido* com a *flag* de *Ping* ativa a cada um dos *Fast File Server* registados.

Se obtiver alguma resposta num determinado intervalo de tempo, assume que tudo correu bem e que o Fast File Server continua ativo, caso contrário, assume que o Fast File Server ficou inativo e elimina-o dos registos do Gateway.

- **Classe HttpWorker**

Esta classe é, também como já dito anteriormente, a classe que tratará do pedido feito pelo cliente. Inicialmente começa por fazer o *parsing* do cabeçalho *HTTP* recebido para descobrir qual é o ficheiro pedido. De seguida, ativa-se a função *trataPedido* existente na classe *Controlador* que irá tratar de fazer o pedido do ficheiro aos *Fast File Servers*, preenchendo assim o *PacoteFicheiro* com os *PacoteChunk* que for recebendo.

Após a função *trataPedido* terminar, o *PacoteFicheiro* ou terá um tamanho superior a zero, o que é sinónimo de que tudo correu bem na busca pelos Fast File Servers, ou terá um tamanho nulo, que é sinónimo de que a busca correu mal. Se a busca correu bem, será enviado um cabeçalho *HTTP* com o código: 200 “OK” e enviado o ficheiro, caso contrário, será apenas enviado um cabeçalho *HTTP* com o código de erro 404.

Nota: O *PacoteFicheiro* é uma classe aninhada dentro de *HttpWorker*, que contém unicamente a lista de *PacoteChunk* e uma *ReentrantLock* que irá fazer o controlo de concorrência sobre essa mesma lista.

- **Classe Controlador**

Esta classe é também uma das principais componentes do Gateway. Tem como principal função a *trataPedido*, que começa por fazer a procura por todos os *Fast File Servers* que estão a servir o ficheiro pedido pelo cliente. Após obter a lista com os dados dos respetivos Fast File Servers, verifica primeiramente se algum está a servir o ficheiro, caso não, retorna imediatamente, marcando o seu insucesso.

Caso contrário, verifica de seguida se só há unicamente um *Fast File Server* a servir o ficheiro, caso sim, é acionada uma busca apenas por **um** *Fast File Server*, o que resulta diretamente na ativação da função *run* da classe *GatewayWorkerFFS* que irá tratar da interação com o *Fast File Server* que tem o ficheiro.

Caso haja mais do que um *Fast File Server* a servir o ficheiro solicitado, é acionada uma busca por **vários** *Fast File Servers*, sendo agora o processo bastante diferente.

Portanto, no caso de uma busca por vários Fast File Servers, precisamos de saber qual o tamanho do ficheiro pedido, para assim poder dividi-lo em blocos que serão repartidos pelos Fast File Servers que o tem. É então feito inicialmente um *PacotePedido* do tipo *metaDados* e envia este a um dos *Fast File Servers* que possui o ficheiro. Como resposta, irá receber um

PacoteResposta com o tamanho em bytes do ficheiro solicitado, pelo que, em seguida será feita a partição do tamanho do ficheiro pelo número do Fast File Servers que tem o ficheiro requerido, e de seguida, é criada uma thread GatewayWorkerFFS para interagir com cada um dos Fast File Servers.

Nota: A classe GatewayWorkerFFS irá receber o intervalo em bytes da parte do ficheiro que tem de pedir ao Fast File Server. No caso da busca apenas por um Fast File Server esse intervalo será nulo, pelo que o Fast File Server, irá saber que tem de enviar o ficheiro todo. No caso de uma busca por vários Fast File Servers o intervalo terá um valor positivo, pelo que o Fast File Server, saberá que tem de enviar apenas o intervalo de bytes especificado.

- **Classe GatewayWorkerFFS**

Esta classe trata exclusivamente de fazer o pedido do ficheiro ao Fast File Server e receber os PacoteChunks por ele enviados. Portanto, primeiramente irá ser preparado o PacotePedido com os dados relativos ao ficheiro que pretende, como o nome, e o intervalo em bytes.

Após preparado, esse pacote é enviado e entra num ciclo infinito onde irá esperar constantemente por PacoteChunks, até que chegue aquele com a flag ultimo ativa. Cada PacoteChunk recebido é colocado em PacoteFicheiro fazendo uso do ReentranteLock por ele disponibilizado.

- **Classe RegistosFFS**

Esta classe encarrega-se de guardar as informações relativas aos registos do *Fast File Servers* no *Gateway*, registos esses guardados sob a forma de um *Map* em *Java* tendo como chave o Tupulo do Fast File Server, e como conteúdo uma lista de Strings, strings essas que são precisamente o nome dos ficheiros que o Fast File Server está a servir.

2.2. Fast File Server

Fast File Server será então a componente que tratará de enviar toda a informação dos ficheiros que está a servir para o *Gateway*.

- **Classe *FastFileServer***

De forma semelhante ao *Gateway*, esta classe é a classe central relacionada ao componente *Fast File Server*. Quando esta se inicia, começa por colocar num *Map* o nome dos ficheiros que esta a servir assim como o espaço em bytes que estes contêm.

De seguida, cria um pacote de registo, que envia ao *Gateway*, e entra num ciclo infinito, onde irá esperar por *PacotePedidos* e responder devidamente de acordo com o pretendido. Se a flag *metaDados* estiver ativa, criará um pacote de resposta, com o tamanho do ficheiro pedido no *PacotePedido*, e enviará este para o *Gateway*. Se a flag *ping* estiver ativa, não terá qualquer reação sem ser a resposta dada por parte da conexão fiável, no entanto, explicaremos isso de forma mais detalhada no tópico Se nenhuma das *flags* estiver ativa, o *PacotePedido* tratara-se então do pedido de um ficheiro que o *FFS* serve, pelo que este criará imediatamente uma thread *FastFileWorker* para tratar da leitura e envio dos bytes do ficheiro solicitados.

- **Classe *FastFileWorker***

Esta classe primeiramente fará a leitura do tamanho do ficheiro solicitado, de seguida, irá verificar se o ficheiro foi solicitado na sua totalidade, ou apenas um intervalo específico, posteriormente efetuará a leitura do ficheiro e partição deste em blocos de 50 bytes para serem enviados, tal é feito invocando para cada bloco a função *read* pertencente à classe *Dados*, adicionando imediatamente o resultado num *Map*, sendo a chave o número do bloco de 50 bytes lido do ficheiro, e o conteúdo o *array* dos 50 bytes lidos.

Após concluído o processo de leitura dos blocos do intervalo especificado, é invocada a função *send*, que fará o envio dos blocos para o *Gateway*. Para fazer o envio, a função irá fazer uso do *PacoteChunk*, tendo que percorrer o *Map*, e criando para cada bloco nele guardado um *PacoteChunk* onde colocará o *array* de bytes, enviando imediatamente ao *Gateway*.

- **Classe *Dados***

Esta classe é simples tempo apenas a função *read* que vai ler um intervalo de bytes específico de um ficheiro e retornar exatamente esses bytes.

2.3. Classe ConexaoFiavel

Esta classe é criada sempre que se quer enviar um pacote, seja ele de que tipo for. A classe recebe como argumentos o socket por onde deve ser feito o envio, e o pacote que deve ser enviado.

Sendo que esta possui duas funções essenciais, que implementam um mecanismo simples para tornar a comunicação fiável entre o *Gateway* e o *Fast File Server* e vice-versa. Existe também uma terceira função que é usada unicamente pelo *Gateway* na receção de *PacoteChunk* enviados pelo *Fast File Server*.

- **Função envia()**

Esta função é usada em qualquer envio de pacotes entre o Gateway <-> FFS. O objetivo desta função é tentar enviar o pacote de forma fiável, ou seja, garantir que o recetor receba o pacote.

Para tal a função irá fazer três tentativas do envio do pacote, sendo que só avança para a seguinte tentativa, se a atual correr mal, ou seja, se algum pacote se perdeu, tanto o pacote que queremos enviar como o pacote de confirmação que o recetor irá enviar. Para cada uma das tentativas, irá simplesmente enviar o pacote pelo socket e criar uma nova thread que vai esperar nesse mesmo socket uma confirmação, ou seja, um *PacoteACK*, e se essa mesma thread não receber o *PacoteACK* assumimos que o envio do pacote correu mal, e vamos para a próxima tentativa fazendo este processo novamente.

Caso o número de tentativas seja excedido, é assumido que algo correu mal na comunicação entre Gateway <-> Fast File Server, e dependendo da situação deve-se adotar comportamentos específicos.

- **Função enviaACK()**

Esta função é usada para enviar o *PacoteACK* ao emissor de um pacote, quando o recetor deseja confirmar o sucesso da transferência. Portanto, simplesmente cria uma *PacoteACK* com o código 1 e envia este pelo socket.

- **Função recebeChunk()**

Esta função é bastante semelhante à função `envia()`, no entanto é usada apenas pelo Gateway, mais concretamente pela classe *GatewayWorkerFFS* no momento da receção de *PacoteChunk* vindos do *Fast File Server*. O objetivo desta classe é prevenir que no momento da receção de *chunks* emitidas pelo *Fast File Server*, se o *FFS* eventualmente deixar de enviar *PacoteChunk*, a thread não fique presa num ciclo infinito, a espera de pacotes que nunca chegarão, sendo necessário posteriormente formar uma resposta do Gateway para o cliente a assinalar a falha.

Daí a função implementar um mecanismo muito parecido com o da função `envia()`, uma vez que esta também vai fazer três tentativas de receção de um pacote, sendo que só avança para a seguinte tentativa, se a atual correr mal, ou seja, se não receber um *PacoteChunk* num determinado intervalo de tempo. Para cada uma das tentativas, irá simplesmente criar uma nova thread que vai esperar pelo *PacoteChunk*, e se essa mesma *thread* não retornar nada durante um determinado tempo, portanto se não receber um *PacoteChunk*, assumimos que o envio deste correu mal e que eventualmente o *PacoteChunk* se perdeu, e avançamos para a segunda tentativa de receção, e, portanto, voltamos a fazer este processo todo novamente. Caso o número de tentativas seja excedido, é assumido que o *Fast File Server* deixou de enviar chunks, e retornamos então um código de erro ao cliente.

3. Especificação dos pacotes usados no protocolo

Antes de partir para a execução concreta do programa resultante, achamos ser necessário perceber cada um dos pacotes partilhados durante a interação do *Gateway* com o diversos *Fast File Servers*.

- **Tupulo**

Esta classe é usada dentro de alguns dos mais importantes pacotes do nosso protocolo, esta classe tem sem dúvida um papel muito importante, pois é, por outra palavras a identidade de uma máquina na rede. Sendo constituído por um endereço que identifica a máquina na rede, e por uma porta que identifica a aplicação.

- **PacotePedido**

O pacote de pedido, é unicamente enviado pelo *Gateway* para *FFS*. Este é constituído por um *Tupulo* de origem e destino, por uma string correspondente ao ficheiro pedido, por duas *flags*, uma relativa aos meta-dados e outra relativa ao *ping*, e por dois inteiros, que ditam o intervalo requerido em bytes do ficheiro.

- **PacoteRegisto**

O pacote de registo, é apenas usado quando enviado pelos *FFS* para o *Gateway*. Este é constituído também por um *Tupulo* de origem e destino, e por uma lista de strings que contém os nomes dos ficheiros que o Fast File Server que enviou o pacote está a servir.

- **PacoteChunk**

O *PacoteChunk*, é enviado pelos *FFS* para o *Gateway* sendo este constituído também por um *Tupulo* de origem e destino, por um array de bytes com conteúdo do ficheiro, por dois inteiros que indicam exatamente o intervalo ao qual pertence o conteúdo do array, e por uma flag último que informa se este é o último de todos os *chunks* pedidos pelo *Gateway*.

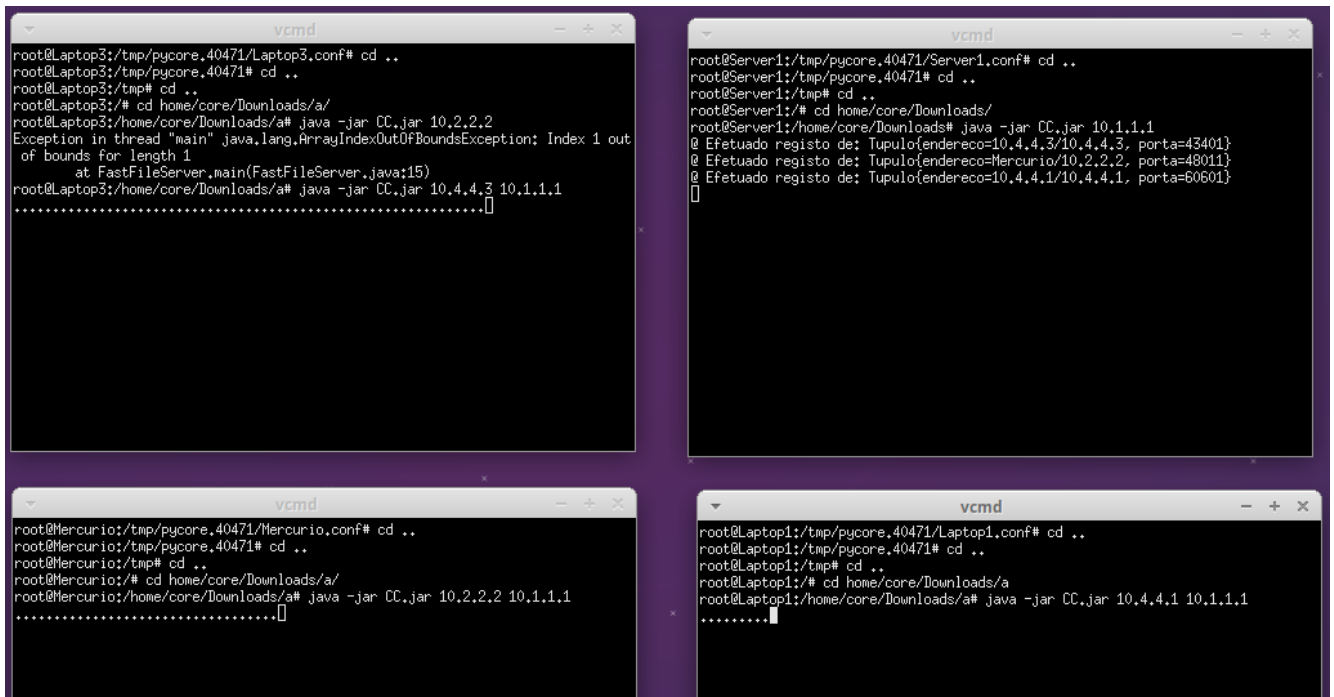
- **PacoteResposta**

O pacote de resposta é exclusivamente enviado pelos *FFS* para o *Gateway* em resposta ao pedido de meta dados. Este é constituído unicamente por um inteiro que contera o tamanho do ficheiro pedido.

- **PacoteACK**

O *PacoteAck* é enviado pelo recetor de qualquer pacote, seja ele enviado do *FFS* para o *Gateway* ou o inverso. É constituído apenas por um inteiro que guardará um código, usado unicamente para controlo de erros.

4. Demonstração do funcionamento



The figure displays four terminal windows, each titled 'vcmd', showing the execution of a Java application across different virtual machines. The top-left terminal (Laptop3) shows an exception: 'Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1'. The top-right terminal (Server1) shows successful registration of three tuples. The bottom-left terminal (Mercurio) shows the execution of the Java application. The bottom-right terminal (Laptop1) shows the execution of the Java application.

```
root@Laptop3:/tmp/pycore.40471/Laptop3.conf# cd ..
root@Laptop3:/tmp/pycore.40471# cd ..
root@Laptop3:/tmp# cd ..
root@Laptop3:/# cd home/core/Downloads/a/
root@Laptop3:/home/core/Downloads/a# java -jar CC.jar 10.2.2.2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 1 out
of bounds for length 1
    at FastFileServer.main(FastFileServer.java:15)
root@Laptop3:/home/core/Downloads/a# java -jar CC.jar 10.4.4.3 10.1.1.1
.....

root@Server1:/tmp/pycore.40471/Server1.conf# cd ..
root@Server1:/tmp/pycore.40471# cd ..
root@Server1:/tmp# cd ..
root@Server1:/# cd home/core/Downloads/
root@Server1:/home/core/Downloads# java -jar CC.jar 10.1.1.1
@ Efetuado registo de: Tupulo{endereço=10.4.4.3/10.4.4.3, porta=43401}
@ Efetuado registo de: Tupulo{endereço=Mercurio/10.2.2.2, porta=48011}
@ Efetuado registo de: Tupulo{endereço=10.4.4.1/10.4.4.1, porta=60601}
[]

root@Mercurio:/tmp/pycore.40471/Mercurio.conf# cd ..
root@Mercurio:/tmp/pycore.40471# cd ..
root@Mercurio:/tmp# cd ..
root@Mercurio:/# cd home/core/Downloads/a/
root@Mercurio:/home/core/Downloads/a# java -jar CC.jar 10.2.2.2 10.1.1.1
.....

root@Laptop1:/tmp/pycore.40471/Laptop1.conf# cd ..
root@Laptop1:/tmp/pycore.40471# cd ..
root@Laptop1:/tmp# cd ..
root@Laptop1:/# cd home/core/Downloads/a/
root@Laptop1:/home/core/Downloads/a# java -jar CC.jar 10.4.4.1 10.1.1.1
.....
```

Figura 1

Na figura 1, são apresentados quatro terminais, sendo estes executados na Máquina Virtual usada nas aulas de Comunicação de Computadores.

O terminal do canto superior está a correr o *Gateway* na entidade *Sevidor1* e os restantes são executados como diferentes *Fast File Servers*, um no *Laptop3* outro no *Mercúrio* e outro no *Laptop1*.

Podemos claramente ver que o *Gateway* registou os dados de cada um dos *FFS*, e podemos ver que os *Fast File Servers* estão a receber pings do *Gateway* (através da escrita de "." nos terminais)

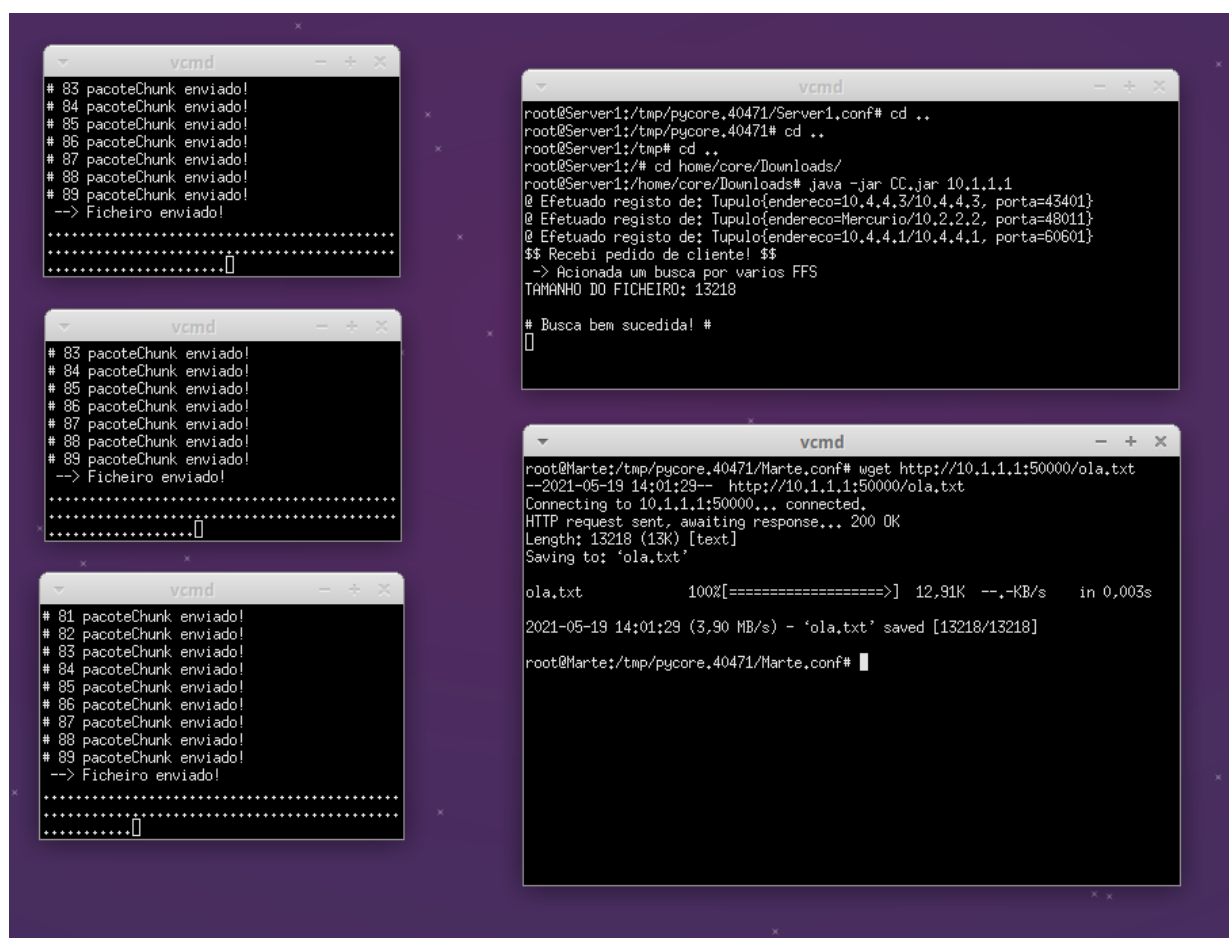


Figura 2

Na figura 2, são novamente apresentados os mesmos quatro terminais da figura anterior, mas após estes terem respondido com sucesso a um pedido do ficheiro chamado “ola.txt” feito por um cliente através de *Marte*, utilizando o comando “wget <http://10.1.1.1:50000/ola.txt>”.

Portanto, como se colocou todos os *Fast File Servers* a servir o mesmo ficheiro, é possível reparar que o tamanho do ficheiro “ola.txt” foi partido em blocos e pedido a cada um dos *Fast File Servers* que o estão a servir.

5. Conclusões e trabalho futuro

Este trabalho prático foi um bastante diferente daqueles que estávamos habituados a desenvolver na Unidade Curricular, tendo acabando por se tornar um dos trabalhos mais interessantes feitos pela equipa até agora. Consideramos que com a sua realização obtivemos bastante conhecimento sobre o que são e como funcionam, de uma forma bastante técnica, as comunicações entre os servidores tanto front-end como back-end.

Assim, a elaboração deste trabalho prático revelou ser de extrema importância para toda a equipa, uma vez que permitiu pôr em prática conhecimentos adquiridos durante as aulas e, ao mesmo tempo, aperfeiçoar e aprofundar outros conhecimentos e ferramentas das quais tiramos proveito, como por exemplo, a linguagem de programação Java e o emulador core. Enriquecendo-nos com uma experiência que certamente será bastante útil no futuro.

Concluindo, apesar de termos uma forma muito pouco eficiente para a recuperação das perdas de pacotes, achamos que o conjunto, como um todo, vai ao encontro daquilo que era esperado, na medida em que nos empenhamos da melhor maneira possível.