Universidade do Minho
Escola de Engenharia

Comunicações por Computador

TP2: Gateway Aplicacional e Balanceador de Carga sofisticado para HTTP

Guilherme da Silva Amorim Martins A89532
José Pedro Carvalho Costa A89519
Simão Paulo da Gama Castel-Branco e Brito A89482

A89532

A89519

A89482

22 de maio de 2021

1 Introdução

Este relatório é relativo ao trabalho prático n.º 2 proposto na Unidade Curricular de Comunicações por Computador com o tema *Gateway Aplicacional e Balanceador de Carga sofisticado para HTTP* que desenvolvemos ao longo de 6 semanas.

O principal objetivo deste projeto é implementar um *gateway* de aplicação que opere exclusivamente com o protocolo HTTP/1.1 e que seja capaz de responder a múltiplos pedidos em simultâneo recorrendo a uma pool dinâmica de N servidores de alto desempenho e usando um protocolo para o efeito. Estes pedidos serão pedidos HTTP GET simples como texto ou imagens.

Numa primeira fase, teremos que desenhar e testar o protocolo *FS Chunk* a funcionar sobre UDP que iremos implementar.

Depois, numa segunda fase, iremos implementar o gateway de aplicação *HttpGw* que irá aceitar os pedidos HTTP GET, fazendo *parsing* do pedido e devolvendo uma resposta HTTP RESPONSE.

De seguida, iremos descrever e explicar todo o processo e mecanismo que tivemos de implementar e percorrer até chegarmos ao resultado final, assim como todas as decisões que fomos obrigados a tomar ao longo da sua execução.

2 Arquitetura da solução

Antes de avançarmos para o desenvolvimento do problema, é importante analisar a arquitetura proposta e perceber tudo aquilo que iremos implementar. A seguinte figura representa, resumidamente, aquilo que nos guiamos para idealizar o nosso projeto, tendo nós, mais tarde, de acrescentar mais algumas funcionalidades que resultarão noutra arquitetura que já a seguir apresentaremos.

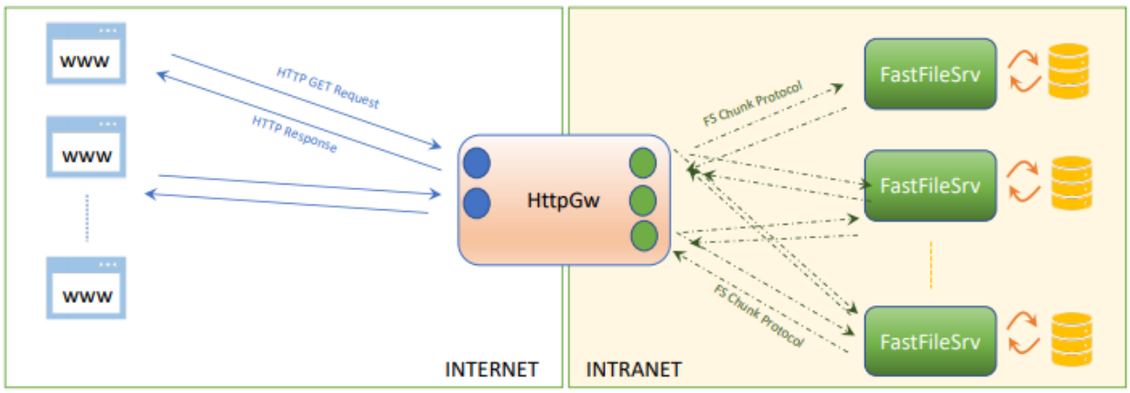

Figure 1: Esquema geral de funcionamento

3 Especificação do protocolo

3.1 Formato das mensagens protocolares e interação

No desenvolvimento deste projeto foi necessário criar um protocolo UDP para a comunicação entre o HttpGW e os FastFileServers. A nosso ver, este protocolo deveria suportar várias operações, como por exemplo subscrição, cancelar a subscrição, cada um destes com a sua password, comunicação do tamanho e da metadata de um ficheiro e envio de um chunk de um ficheiro.

O pacote do protocolo UDP foi desenhado de forma a não ultrapassar os 512Bytes, pois deste modo não irá ser fragmentado na rede, sendo as possíveis retransmissões menos custosas.

Para tal inserimos no cabeçalho do pacote um primeiro inteiro para a identificação da operação que está a ser realizada, sendo o 1 para uma operação de subscrição, o 2 para o cancelamento de subscrição, o 3 para um pedido do tamanho de um ficheiro e 4 para um pedido de chunk de um ficheiro.

Um segundo inteiro para identificar o id de sessão do cliente que está a originar os pedidos. Será usado para guardar a informação das operações 3 e 4 no sítio certo. No caso de ser subscrição ou cancelamento de subscrição este campo não tem nenhum valor em específico.

Um terceiro inteiro com dois significados, dependendo da operação. No caso de ser um pedido de tamanho do ficheiro, irá lá ser guardado o tamanho do ficheiro em Bytes. Se for um pedido de chunk, este inteiro contem o id do chunk a enviar, sendo o offset do chunk = id do chunk * MAXCHUNKSIZE.

Por último, o cabeçalho tem um array de Bytes com um máximo de 400 Bytes, e três significados distintos, se for uma operação de subscrição ou cancelamento de subscrição, o campo contem os bytes da password, sendo a password de subscrição a string "Subscribe" concatenada com a string com o valor do ip do FastFileServer. É feito um hash com esta string, usando SHA-256. O cancelamento da subscrição é realizada da mesma forma, trocando a string "Subscribe" com a string "Unsubscribe". Foi feito desta maneira para que a password seja diferente entre cada FastFileServer, não sendo possível simplesmente copiar o pacote de subscrição de uma máquina diferente, caso existisse algum atacante que se quisesse subscrever na rede através de fazer sniffing à mesma. Se a operação for um pedido de tamanho, o array terá os Bytes de uma string que contem a metadata do ficheiro, mais especificamente o tipo de ficheiro, a data de última modificação e o tamanho (cada campo é concatenado com uma string para colocar no cabeçalho HTTP Response). Finalmente, este array de Bytes contem os bytes do chunk, se for o caso do requisito de chunk.

```
public class PacketUDP implements Serializable{
    private int packettype; //1-Subscribe|2-Unsubscribe|3-Size|4-Chunk
    private int packetid;   //ID do packet = ID da sessão TCP -> Necessário para guardar o Chunk no sitio certo no map
    private int chunkid;    //Se for um chunk vai ter de ter identificacao do chunk
                            //Se for um pedido de size vai ter o tamanho do ficheiro -> -1 se mão existe
    private byte[] chunk;   //Chunk ou Metadata(String) ou password
```

Figure 2: Codigo do pacote FSChunk utilizado sobre UDP

4 Implementação

Após uma análise afincada do problema, optamos por realizar o projeto na linguagem de programação Java, visto que concluímos que seria a linguagem mais confortável para todos os elementos do grupo.

Depois disso, estabelecemos uma arquitetura diferente da proposta no enunciado para que nos facilitasse na altura da implementação. O diagrama da nossa arquitetura a que chegamos no final foi este:

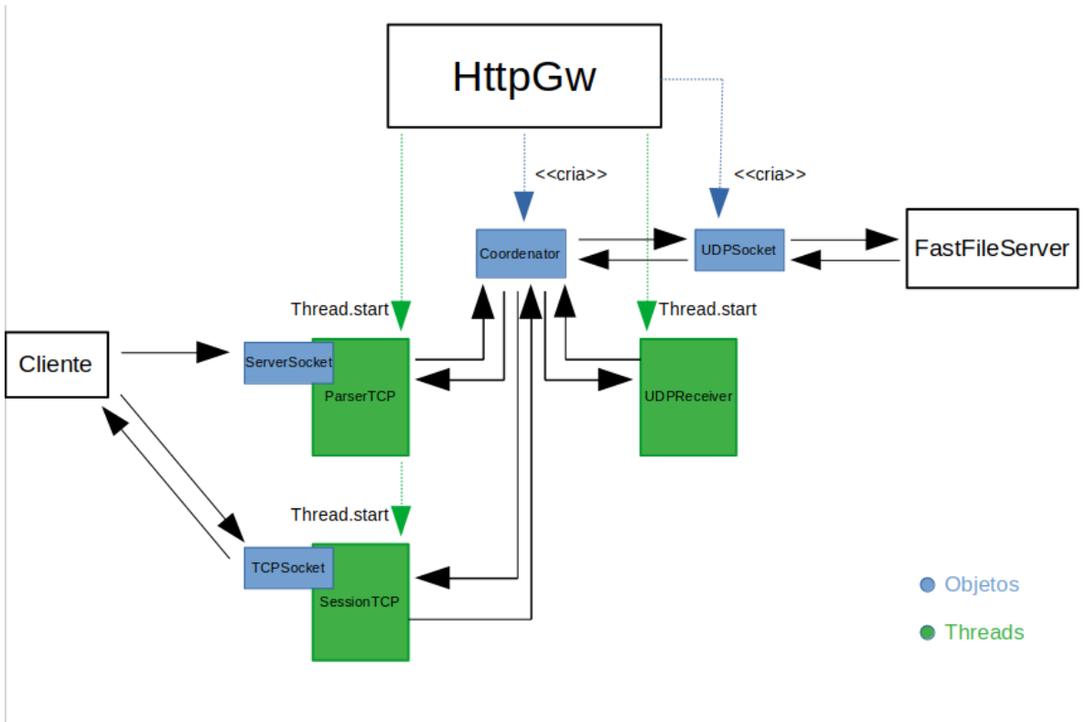

Figure 3: A arquitetura do nosso programa

4.1 HttpGw

Consideramos o gateway aplicacional o processo programa principal, sendo que é este que faz a conexão entre o cliente e os servidores que contêm os ficheiros, e como tal, é de extrema importância.

A primeira coisa que o programa faz é obter o ip da máquina automaticamente, com a biblioteca java.net.NetworkInterface (ou pode também recebe-lo pela linha de comandos ao ser executado) e criar um DatagramSocket para a comunicação sobre UDP com os FastFileServers. Sendo que este socket, tal como muita informação, irá ter de ser partilhada entre diferentes threads, é criado um objeto do tipo CoordinatorHttpGw, do qual iremos falar mais à frente. Por fim, nesta thread principal, são criadas as threads ParserTCP, à qual passamos os objeto CoordinatorHttpGw e o ip da máquina e a thread ReceiverUDP ao qual passamos também o objeto CoordinatorHttpGw.

4.1.1 ReceiverUDP

A thread ReceiverUDP encarrega-se de receber todos os pacotes vindos dos FastFileServers, verificar se o servidor que enviou e a informação estão válidos, e armazenar a informação no objeto CoordinatorHttpGw. Isto é feito com um ciclo while, e sempre que recebe um pacote pelo socket UDP guarda-o na variavel p1.

De seguida, consoante a operação, realiza ações diferentes. Se for um pedido de subcrição, caso o servidor ainda não esteja inscrito e tenha a password válida, é registado numa tabela de servidores contida no CoordinatorHttpGw.

Se o servidor estiver registado, for um pacote de tamanho e o tamanho no ChunkManager (objeto falado mais a frente) no mapa de ChunkManagers localizado no CoordinatorHttpGw com a chave igual ao id da sessão do cliente estiver a vazio, o tamanho é atualizado e a metadata é inserida.

No caso do servidor estar registado e ser uma operação de chunk e o chunk no mapa de chunks com a chave igual ao id do chunk, que está no mapa de ChunkManagers com a chave igual a sessão dentro do CoordinatorHttpGw, for null, insere lá o chunk.

Finalmente, estando o servidor registado e sendo uma operação de cancelamento de subscrição, o processamento assemelha-se ao de subscrição, onde é verificada se a password está correta, e se estiver, o servidor é retirado do mapa de servidores.

4.1.2 ParserTCP

O ParserTCP é criado o ServerSocket para a receção de comunicação sobre TCP. Sempre que recebe um pedido cria um socket e caso o número de threads máxima para SessõesTCP seja maior que zero cria uma thread SessionTCP onde passa o socket e o CoordinatorHttpGw. Caso não possa criar a thread, é feito await, para esperar pela sua vez de criar a SessionTCP.

4.1.3 SessionTCP

Esta classe encarrega-se de comunicar com o cliente e requerir tudo o que é necesário para a transmissão do ficheiro. Para tal, contêm algumas constantes, como TRIES, o numero de re-envios de pedidos aos FastFileServers, o RoudTripTime, sendo o tempo esperado para o envio e receção de um pacte e o RequestIncrement, que consideramos um incremento de tempo esperado por cada tentativa de pedido de informação, sendo estes dois ultimos em milisegundos. Para a comunicação com o cliente, pede um id ao CoordinatorHttpGw para identificar a thread e valida o pedido Http do cliente.

Se o pedido estiver inválido, envia a resposta ao cliente com o código 400 Bad Request. Se o pedido estiver válido mas não for GET, envia uma resposta com o código 501 Not Implemented.

Caso esteja válido e seja um pedido GET, primeiro pede o tamanho do ficheiro a um FastFileServer. Se o tamanho for válido, significa que o ficheiro existe, passando para o pedido dos chunks. Estes pedidos aos FastFileServers são feitos através do CoordinatorHttpGw, pegando num servidor aleatório que esteja no mapa de FastFileServers e envia-lhe o pacote UDP. O pedido é feito a um servidor aleatório pois foi considerado que todos os servidores têm acesso ao mesmo ficheiro, tal como era descrito no enunciado. Estes pedidos são feitos dentro de um ciclo, sendo pedidos todos os pacotes requeridos (no caso do tamanho é só feito um pedido, no caso dos chunks são feitos tantos pedidos quantos chunks necessários), sendo feito de seguida um sleep à thread, usando as constantes de tempo anteriormente definidas. Quando a thread acorda é verificado se foram recebidos os pacotes, se não foram recebidos faz um re-envio dos pedidos, entrando aqui a constante TRIES. Se ao fim das tentativas os pacotes necessários não chegarem, a thread desiste. Se desistir durante o pedido de tamanho envia uma resposta com o código 404 Not Found, e com conteudo e um ficheiro html que se encontra na variavel filenotfound. Se desistir durante o pedido dos chunks envia o códgo 500 Internal Server Error. Se depois do sleep os pacotes tiverem sido todos recebidos, é enviada a resposta com o código 200 OK, a informação do servidor e a data, a metadata do ficheiro, informação a dizer que a conexão será fechada, e os chunks do ficheiro.

Se não existir nenhum FastFileServer conectado é enviado uma resposta com o código 503 Service Unavailable. Se houver algum erro na thread é enviado o código 500 Internal Server Error.

Por fim é feito o flush da informação no socket, fechado o mesmo e aumentado o número de threads máxima que podem ser criadas.

4.1.4 CoordinatorHttpGw

O CoordinatorHttpGw é um objeto muito importante e muito usado pelo HttpGW. É usado desde para guardar informação, até controlo de concorrência.

Em primeiro lugar é armazenado o número máximo de Bytes que um chunk pode ter, já falado anteriormente.

Há uma varivel onde se guarda o número máximo de threads que podem ser criadas para sessões TCP em conjunto com um lock e uma variavel condicional. Estas variaveis são usadas para o controlo de concorrência e também para que o servidor não fique sobrecarregado, e em caso de um ataque DOS possa sobreviver.

Guarda-se o soket UDP, também com o seu lock, pois apesar de apenas existir uma thread capaz de ler do socket, todas as SessionTCP vão ter acesso ao socket para escrita, sendo necesário existir uma variavel lock.

É também guardado um mapa com os FastFileServers conectados, em que a chave é a concatenação do ip e da porta do servidor e o valor é um objeto FFSInfo, que apenas contém o endereço ip e a porta, aliado a um lock e a um inteiro que guarda o número de servidores conectados. Tal como em cima, esta tabela vai ser usada por varias threads, sendo necessário o lock.

É ainda guardado um inteiro com o próximo número a ser dado a uma sessão TCP, com um lock, pela mesma razão em cima, e o número máximo que a variável requestID pode ter.

Por fim, é guardado um mapa de ChunkManagers, sendo a chave o id de sessão TCP e o valor um objeto ChunkManager. Este objeto serve para guardar tudo o que é necessário para os pedidos aos FastFileServers e a resposta ao cliente: o nome do ficheiro, a metadata do ficheiro, o tamanho do ficheiro, e os bytes do ficheiro, sendo estes guardados num mapa em que a chave é o id do chunk.

4.2 FastFileServer

O FastFileServer é também considerado por nós um programa de extrema importancia, sendo que sem ele o gateway apicacional não irá retornar nenhum ficheiro. Cada FastFileServer tem uma tabela de ficheiros que pode enviar, sendo que alguns deles podem não ser para envio.

Na execução do FastFileServer são passados como argumentos o ip e a porta do HttpGw para ser realizada a conexão UDP. O programa descobre o ip dá máquina, que é feito da mesma maneira que no HttpGw, e cria o socketUDP. De seguida envia um pacote de subscrição e cria uma thread para o envio de re-subscrições. A thread principal e esta comunicam através do objeto FFSCoordinator, que contem uma variável boleana que começa a falso, e um lock. Sempre que a thread principal envia um pacote ao HttpGw coloca a variavel boleana a falso. Na thread de re-subscrição é feito um ciclo while, e de 1 em 1 minuto, sendo este 1 uma constante do programa, é verificado o valor da variavel boleana do FFSCoordinator. Se estiver a verdadeiro é reenviado um pacote de subscrição. Depois desta avaliação, a variavel booleana é posta a true e a thread volta ao inicio do ciclo.

Depois da thread principal criar a thread de re-subscrição, cria um pacote de cancelamento de subscrição e cria um ShutdownHook, através do import da class java.lang.Runtime, passando-lhe um thread Unsubscribe, que simplesmente envia o pacote de cancelamento de subscrição ao HttpGw.

De seguida, é feito um ciclo while, onde se recebe um pacote UDP. Se o pacote não tiver o ip de quem o enviou igual ao ip do servidor, o pacote é ignorado.

Se for um pacote do tipo 3 e o ficheiro que foi pedido não consta da libraria do FastFileServer, é enviado o pacote que foi recebido com o campo chunkid, que representa o tamanho do ficheiro, igual -1. Se o ficheiro constar da libraria, o servidor vai buscar o tamanho através da class import java.nio.file.Files, a metadata relativa a data de ultima modificação e o tamanho do ficheiro com a class java.nio.file.attribute.BasicFileAttributes, sendo usada a class java.text.SimpleDateFormat para converter a data, e por fim o tipo MIME do ficheiro com a class java.io.File, usando o objeto FileNameMap. Temos duas maneiras distintas de obter o tamanho do ficheiro pois foram feitas em alturas distintas, sendo o requisito do tamanho do ficheiro feito logo no inicio do trabalho, e requisito da metadata do ficheiro só no fim.

Finalmente, se for um pacote do tipo 4, coloca-se na variavel chunk do pacote o chunk que foi pedido, sendo obtido através do nome do ficheiro, que vinha no campo chunk, e o chunk id, que em conjunto com o tamanho de um chunk, obtém se o offset do ficheiro. Para a leitura utilizamos a class java.io.RandomAccessFile.

5 Testes e resultados

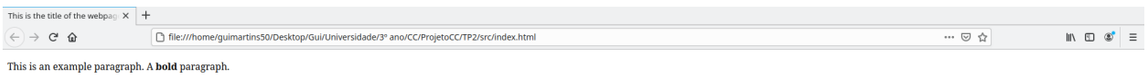

Figure 4: Ficheiro html que pretendemos descarregar

```
guimartins50@guimartins50-HP-Laptop-15-dw2xxx:~/Desktop/Gui/Universidade/3º ano/CC/ProjetoCC/TP2/src$ java HttpGw
HttpGw's IP: 192.168.1.3
HttpGw's TCPPort: 8080
HttpGw's UDPPort: 8888
        FFS subscribed
Number of servers: 1

My Session ID: 0
Http Method: GET
File: index.html
Saiu do ciclo. Size=193
Requested Chunks
40
Requested Chunks
Download Complete!
Last-Modified: Mon, 24 May 2021 18:59:59 WEST
Content-Length: 193
Content-Type: text/html

Http Response sent
```

Figure 5: Gateway

```
guimartins50@guimartins50-HP-Laptop-15-dw2xxx:~/Desktop/Gui/Universidade/3º ano/CC/ProjetoCC/TP2/src$ java FastFileS
erver 192.168.1.3 8888
My ip: 192.168.1.3

Server ip: 192.168.1.3
Server port: 8888

Package Subscribe Sent
->Send filesize+metadata
->Send chunk from index.html starting from0
```

Figure 6: FFS

```
guimartins50@guimartins50-HP-Laptop-15-dw2xxx:~/Desktop/Gui/Universidade/3º ano/CC/ProjetoCC/TP2/src$ wget 192.168.1
.3:8080/index.html
--2021-05-25 20:21:17--  http://192.168.1.3:8080/index.html
Connecting to 192.168.1.3:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 193 [text/html]
Saving to: ‘index.html.1’

index.html.1                100%[=========================================>]     193  --.-KB/s    in 0s

Last-modified header invalid -- time-stamp ignored.
2021-05-25 20:21:17 (37,4 MB/s) - ‘index.html.1’ saved [193/193]
```

Figure 7: Wget do ficheiro index.html

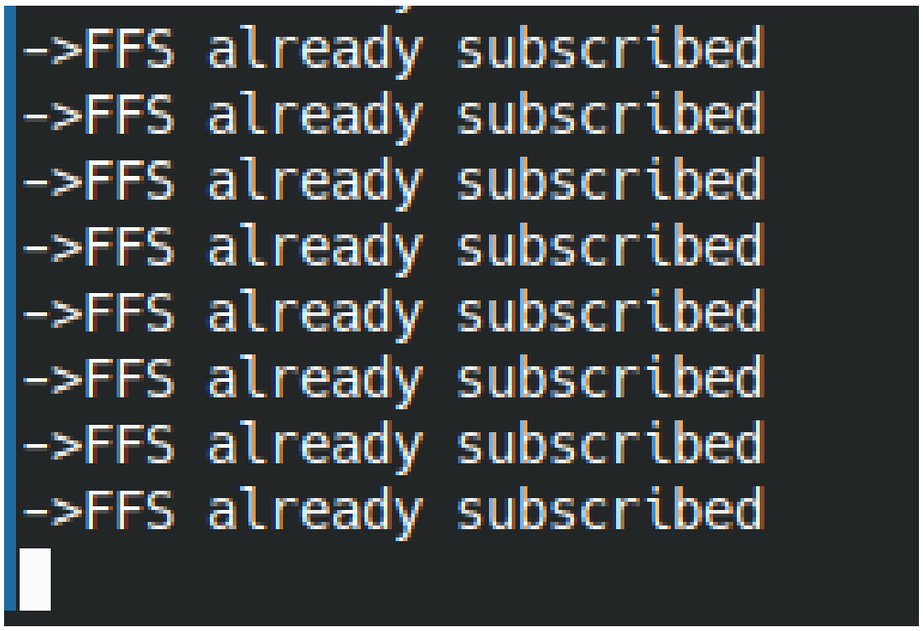

Figure 8: Comportamento do HttpGw após determinado tempo com ausência de pedidos

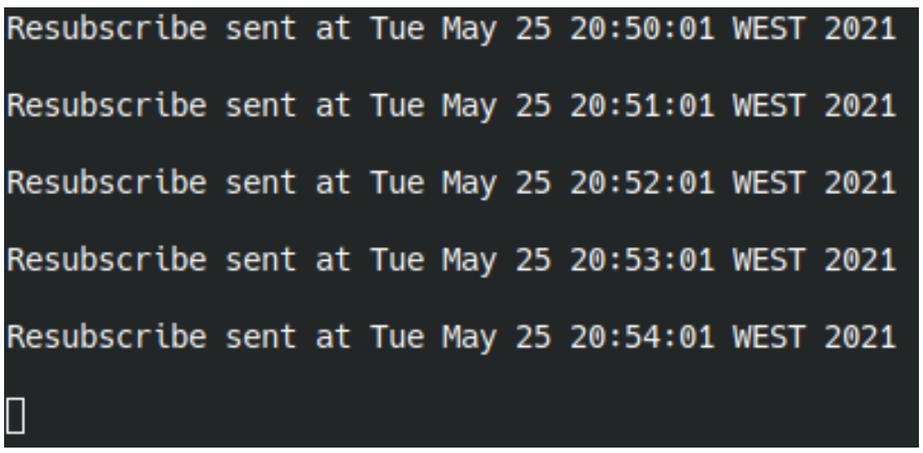

Figure 9: Comportamento do FFS após determinado tempo com ausência de pedidos

Nas seguintes imagens podemos ver o inicio e o fim do ultimo teste feito no core, o qual pensamos que demonstra a rsistencia do trabalho. Iniciamos apenas um FastFileServer no Pico, sendo que este local tem perdas para a rede, e fizemos dois pedidos em paralelo de ficheiros relativamente grandas, aproximandamente 9MegaBytes, sendo que o HttpGw conseguiu dar resposta a ambos.

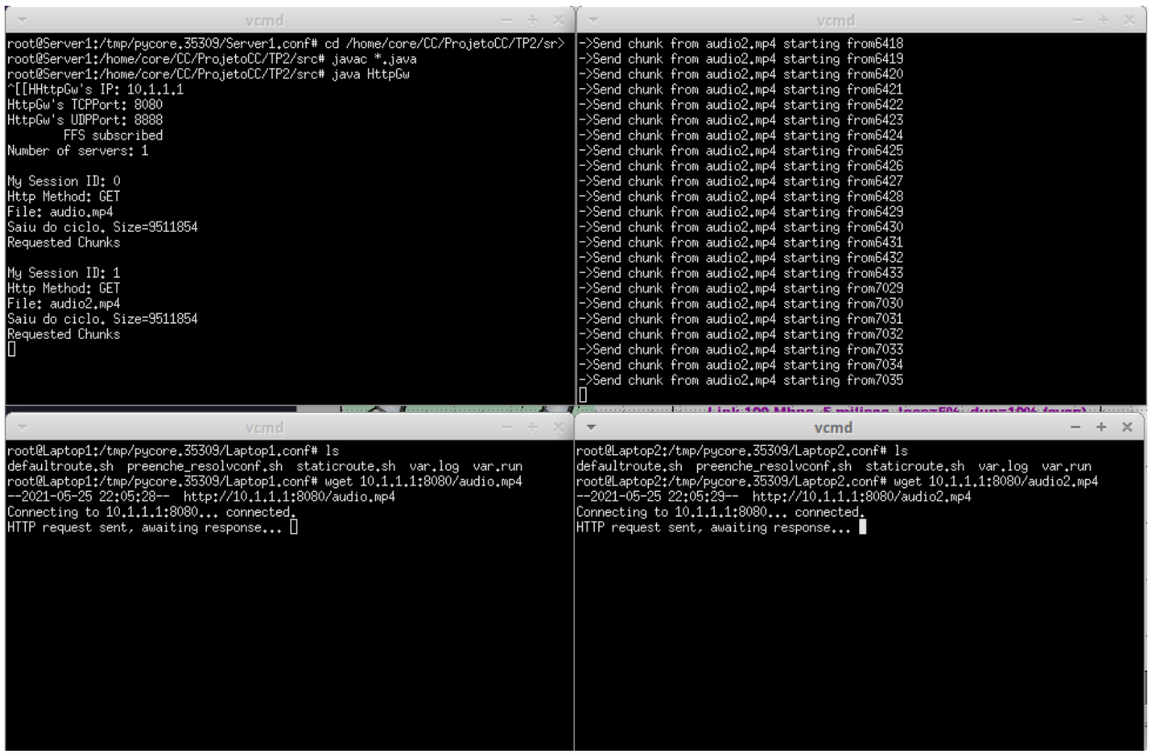

Figure 10: Inicio do teste no core

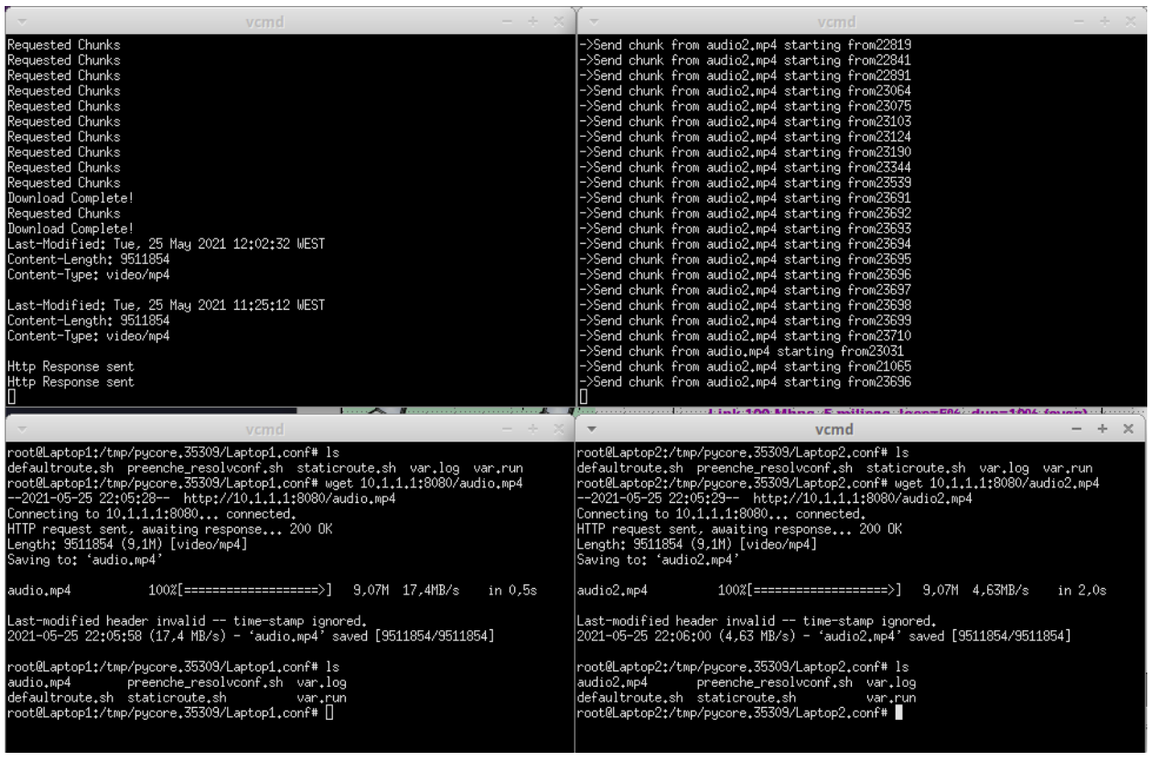

Figure 11: Fim do teste no core

6 Conclusões e trabalho futuro

Este trabalho foi completamente diferente daqueles que estávamos habituados a realizar no contexto desta Unidade Curricular e também muito diferente dos que realizamos em Redes de Computadores, no entanto, não foi motivo para desmotivar o grupo. Pelo contrário, o grupo sentiu que foi um projeto apelativo e um verdadeiro trabalho de pura engenharia.

Durante a elaboração do projeto tivemos de enfrentar diversas dificuldades que conseguimos ultrapassar depois de muita dedicação, conduzindo-nos assim a tomar decisões importantes para o rumo do trabalho.

Existe alguns aspetos que o grupo considera que podia ser melhorado que é, por exemplo, o caso do chunk id ser um int, visto que, se fosse do tipo long, o tamanho máximo de um ficheiro requerido poderia ser muito maior. Sentimos que podíamos ter melhorado na segurança protegendo contra ataques DOS, ignorando o pedido por completo, em vez de só ter um numero máximo de threads. Poderia se ter criado uma tabela de ips que fazem pedidos e ignorar ips com muitos pedidos por um determinado período de tempo. Gostavamos também de ter implementado uma funcionalidade para que o HttpGw pudesse dar unsubscribe aos servidores FFS que não tiverem a responder e guardar o numero de pedidos ao FFS e o numero de respostas. Pensamos em ter um pacote para descobrir o RTT, em vez de ser fixo. O pacote de id 5 que enviava o tempo atual de envio e recebia o mesmo pacote com o tempo de envio e calculava o RTT. Pretendíamos suportar pedidos persistentes, esperando por mais pedidos em vez de fechar o socket. E, por último, seria interessante suportar Http Head request, que seria de fácil implementação, sendo que seria verificar se a metadata do ficheiro requerido está correta.

Conseguimos, na realização deste trabalho prático, aperfeiçoar todos os conhecimentos aprendidos nas aulas teóricas sobre *Http Requests* e *Http Responses* com conexões TCP e UDP através de sockets e ajudou-nos a aprofundar outros conhecimentos e ferramentas que tiramos proveito, tal como a linguagem de programação Java, desenvolvendo assim o nosso à vontade com a linguagem que acreditamos ser uma mais-valia no futuro.

De um modo geral, o grupo faz uma avaliação muito positiva do trabalho desenvolvido, visto que vai de encontro aquilo que era esperado e porque o grupo sente que se empenhou da melhor maneira possível.