Relatório TP2 - Grupo 1 do PL2(G21)

Nuno Aguiar[1 [A100480]]

Universidade do Minho, Braga, PT https://www.uminho.pt/PT

Resumo O presente documento descreve a nossa solução do problema proposto no TP2 da UC de Comunicações por Computadores da Licenciatura em Engenharia Informática da Universidade do Minho. Neste são referidos vários tópicos importantes sobre a realização do trabalho, assim como as decisões tomadas ao longo do seu desenvolvimento.

Keywords: FS_Node · FS_Tracker · Weighted Round Robin · Peso · TCP · UDP.

1 Introdução

Foi-nos proposto desenvolver um serviço avançado de transferência de arquivos numa rede peer-to-peer(P2P) com múltiplos servidores, os quais também desempenham o papel de clientes no mesmo serviço. Dessa forma, estabelece-se um diálogo entre pares, onde, num dado momento, um arquivo esteja disponível em diversos peers. A transferência pode ocorrer de qualquer um deles, ou mesmo de vários simultaneamente com o objetivo de otimizar a disponibilidade e o desempenho.

Um peer que inicia o download de um arquivo pode imediatamente compartilhá-lo com outros peers, mesmo que ainda esteja incompleto, dividindo-o em blocos identificáveis.

2 Arquitetura da solução

Para ajudar a explicar a arquitetura da solução elaborada, criamos um diagrama para exemplificar as relações entre *FS_Node* e *FS_Tracker* e entre *FS_Node* e *FS_Node*.

FS_Tracker: Servidor que cada vez que um node se conecta a ele cria uma thread para falar com esse Node diretamente via socket TCP, o *FS_Tracker* contêm a informação relativa à localização dos ficheiros.

FS_Node: Cada Node tem duas threads ativas a qualquer momento. Uma delas está sempre a ouvir pedidos de transferência de outros Nodes via UDP (quando recebe um pedido envia-lhe os blocos pedidos do ficheiro requisitado), a segunda thread comunica com o *FS_Tracker* via TCP e pode também requisitar blocos a outros nodos, criando assim uma thread nova para cada node a que vai pedir blocos.

Estruturas auxiliares:

Tracker - Estruturas a que o *FS_Tracker* recorre para executar as funções necessárias. O *ficheiro_do_nodo* é um dicionário que guarda as informações relativas aos ficheiros e aos nodos que contêm esses ficheiros, o *memoriaLogin* é utilizado para quando um *FS_Nodo* com um peso(explicamos o peso na parte da implementação) maior do que 0 ser relembrado para quando voltar a reconectar-se ao *FS_Tracker* continuar com o valor que continha antigamente.

Node - Estruturas a que um *FS_Node* recorre para executar as suas funções. O dicionário *blocos_recebidos* armazena o nome dos ficheiros, o número dos blocos e a informação contida por cada bloco enquanto não forem recebidos todos os blocos de um ficheiro para o poder guardar na pasta vinculada ao *FS*, o *blocos_em_falta* é uma lista que um Node cria cada vez que transfere um ficheiro novo e no caso de não receber um certo bloco coloca-o na lista para no final voltar a pedir os blocos que lhe faltam.

Mensagens Protocolares: Referem-se ao tipo de comunicação transmitida entre o *FS_Node* e o *FS_Tracker*, incluindo o reply, se houver, que a mensagem recebe,bem como as trocas de mensagens entre um *FS_Node* e outro *FS_Node*.

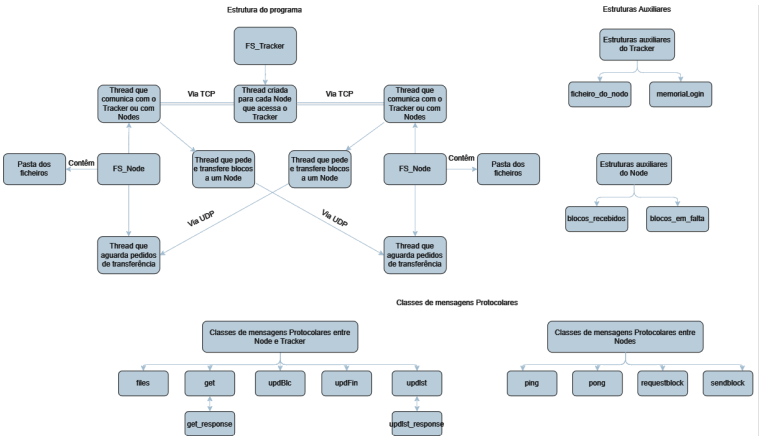

Figura 1. Arquitetura geral da solução

3 Especificação do(s) protocolo(s) propostos

3.1 Formato das mensagens protocolares

Para o desenvolvimento do protocolo começamos pelas mensagens protocolares entre o *FS_Tracker* e o *FS_Node*, estas mensagens são enviadas via **TCP** e

cada mensagem tem uma chave que é por onde o *FS_ Tracker* vai reconhecer que tipo de ação terá de executar.

As mensagens entre o *FS_ Node* e o *FS_ Tracker* têm um tamanho fixo de 1024, pois achamos ser o necessário para enviar qualquer uma destas mensagens sem problemas.

files: Esta mensagem é enviada assim que se dá a conexão do *FS_ Node* ao *FS_ Tracker*. Esta é constituída por 2 segmentos: o primeiro é a chave enviada **files** e o segundo é uma lista de tuples que contém o número total de blocos que um ficheiro tem e o nome do ficheiro sendo tudo enviado no segmento **data**.

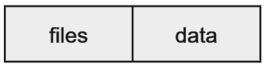

Figura 2. Formato da mensagem files

get: Esta mensagem é enviada pelo *FS_ Node* ao *FS_ Tracker* após o utilizador ter digitado o comando *get filename* sendo então enviada uma mensagem constituída por 2 segmentos: o primeiro é a chave enviada **get** e o segundo é o nome do ficheiro que o *FS_ Node* pretende transferir estando representado com **filename**.

Figura 3. Formato da mensagem get

get_response: A mensagem **get** tem uma reply do *FS_ Tracker* para o *FS_ Node*. Esta mensagem é composta por 4 segmentos: o primeiro é o **número total de blocos do ficheiro** que se pretende transferir, o segundo é uma lista ordenada de tuples a começar no bloco 1 até ao último bloco (cada bloco têm consigo um array que contém os Nodes que possuem esse bloco) (**hosts com blocos**), o terceiro segmento é para se saber quantos Nodes individuais é que possuem pelo menos um bloco do ficheiro (**nº de hosts individuais**) e o último segmento é **End_Transmission**, pois como este é uma das únicas mensagens entre o Tracker e o Node que pode necessetiar de ser enviada repartida, precisando assim de sinalizar o fim da mensagem.

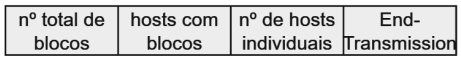

Figura 4. Formato da mensagem get_response

As próximas mensagens são trocadas entre o *FS_Node* e o *FS_Tracker* enquanto ocorre uma transferência ou após a transferência ter acabado.

updBlc: Esta mensagem é enviada pelo *FS_Node* ao *FS_Tracker* cada vez que é concluída a transferência de um bloco pelo Node de forma a notificar o Tracker que o Node atual já possui o bloco sendo então a mensagem constituida por 5 segmentos: o primeiro é a chave enviada **updBlc**, o segundo é o nome do ficheiro ao qual o bloco pertence, o terceiro é o **nº do bloco** transferido, o quarto é o **peso** a adicionar ou a subtrair ao *FS_Node* sendo o seu nome o último segmento **hostname**.

Figura 5. Formato da mensagem updBlc

updlst: Esta mensagem é enviada pelo *FS_Node* ao *FS_Tracker* no caso de ser necessário voltar a pedir blocos, sendo então pedida uma nova lista de Nodos atualizada. A mensagem é constituida por 2 segmentos: o primeiro é a chave enviada **updlst** e o segundo é o nome do ficheiro ao qual é pretendido pedir a nova lista atualizada com os pesos novos dos Nodos.

Figura 6. Formato da mensagem updlst

updlst_response: Esta mensagem é a reply enviada pelo *FS_Tracker* ao *FS_Node* após ter sido pedido a lista de blocos. A mensagem comporta-se de forma semelhante à *get_response* sendo que a diferença é a menor quantidade de segmentos da mensagem: em vez de 4 são apenas 2, um deles a lista de tuples **hosts com blocos** e o outro o segmento final.

End-Transmission.

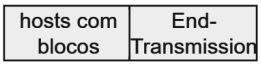

Figura 7. Formato da mensagem updlst_response

updFin: Esta mensagem é enviada após ter sido finalizada a transferência de um ficheiro e este ter sido retirado do dicionário *blocos_recebidos* e passado para um ficheiro na sua pasta, o *FS_Node* notifica o *FS_Tracker* do final com uma mensagem que contém 2 segmentos: o primeiro é a sua chave **updFin** e o segundo é o nome do ficheiro (**filename**). Após ser notificado o *FS_Tracker* atualiza a sua estrutura de memória.

Figura 8. Formato da mensagem updFin

De seguida estão as mensagens que apenas são trocadas entre *FS_Nodes*, sendo estas via **UDP**.

ping: Esta é a primeira mensagem enviada ao *FS_Node* que contém os blocos do ficheiro tendo como confirmar que o *FS_Node* está ativo e **calcular um intervalo de tempo** a ser utilizado depois quando for pedido o bloco com a informação. Esta mensagem apenas contêm um **ping**.

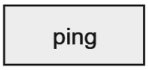

Figura 9. Formato da mensagem ping

pong: Resposta à mensagem *ping* com uma mensagem que contém um **pong**.

Figura 10. Formato da mensagem pong

request_block: Esta mensagem é enviada ao *FS_Node* que contém os blocos do ficheiro que se pretende transferir sendo que a mensagem está dividida em 2 segmentos: o **nº do bloco** requisitado e o nome do ficheiro ao qual o bloco pertence (**filename**).

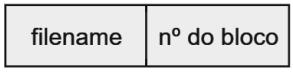

Figura 11. Formato da mensagem request_block

send_block: Após receber o *request_block* é então criada a mensagem que contém o conteúdo do bloco. Cada mensagem têm um total de **1200 bytes** sendo que os primeiros **2 bytes** contêm um método de verificação do ficheiro **checksum**, sendo depois comparado com o **checksum** calculado no *FS_Node* que recebe a informação. Os **4 bytes** seguintes são o **número do bloco** que está a ser enviado e por fim os restantes **1194 bytes** são a informação do ficheiro (**data**).

Figura 12. Formato da mensagem send_block

3.2 Interações

De forma a explicarmos melhor as interações entre os Nodes e o Tracker decidimos desenvolver um diagrama temporal que representaria as interações que ocorreriam desde que um *FS_Node* se conectou ao *FS_Tracker* e começou a descarregar um ficheiro até à conclusão da transferência.

Como é possível observar após um Nodo se conectar ao Tracker é enviada a mensagem *files*. De seguida caso o Nodo pretenda transferir um ficheiro é enviada uma mensagem *get* ao *FS_Tracker* sendo então aguardada a mensagem *get_reply*.

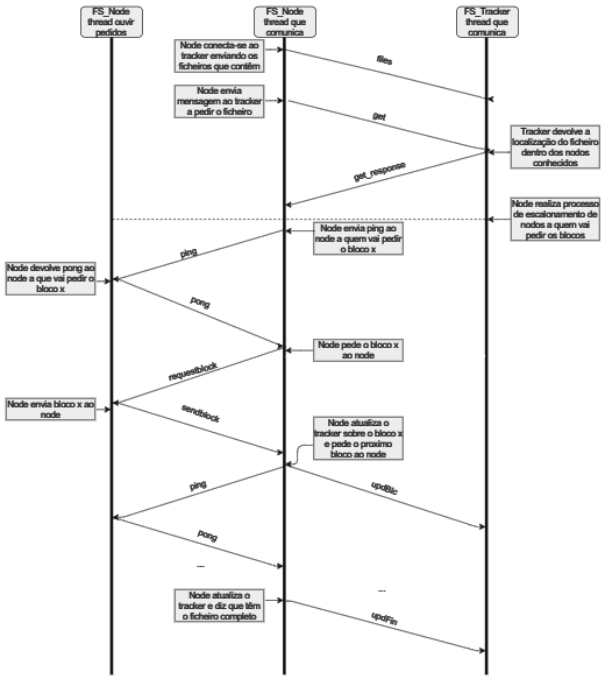

Figura 13. Diagrama Temporal de Protocolos

A partir do momento que o *FS_ Node* têm conhecimento da localização dos blocos é feito então o processo de escalonamento de forma a decidir a maneira mais eficiente de pedir os blocos a todos os Nodos que contêm pelo menos 1 bloco desse ficheiro. Neste diagrama representamos a conversa que aconteceria entre ambos os Nodos, começando por ser enviado um *ping* e é aguardado um *pong*. Com o tempo que demora a enviar e a receber estas duas mensagens é calculado um intervalo de tempo que irá ser utilizado quando se for pedir o bloco. Por fim é enviado o *request_ block* e aguarda-se a mensagem *send_ block*. Assim que esta informação é recebida é então enviada uma mensagem ao *FS_ Tracker* para ele saber que o *FS_ Node* que está a transferir o ficheiro tem mais um bloco na sua posse e enquanto este update é enviado o *FS_ Node* continua a pedir os restantes blocos sendo que este processo se repete até à sua conclusão, onde é enviado pelo *FS_ Node* ao Tracker a mensagem *updFin* que informa o *FS_ Tracker* da conclusão da transferência. As mensagens que não estão aqui representadas são a mensagem *updlst* e a *updlst_ response*. Estas apenas ocorrem no caso de ter ocorrido um erro a receber um ou mais blocos sendo que neste caso após ter sido concluída a primeira fase de transferência será verificado na lista *faltam_ blocos* do *FS_ Node*, que blocos faltam enviar sendo então requisitado ao *FS_ Tracker* uma lista atualizada dos nodos que contêm os blocos e volta-se a repetir o processo de transferência para os blocos que faltam.

4 Implementação

4.1 Detalhes

Para o desenvolvimento do projeto escolhemos a linguagem **python** visto que achamos que era a que sentia-mos mais confortáveis a usar para desenvolver um projeto destes. Desta forma separamos o projeto em 2 fases como recomendado no enunciado, na primeira desenvolvemos a conexão via **TCP** do *FS_ Tracker* com o *FS_ Node*, tendo sido elaborados os protocolos anteriormente. Na segunda fase focamo-nos nas conexões entre Nodos via **UDP** tendo em atenção os constantes updates que o *FS_ Node* teria de enviar ao *FS_ Tracker* sobre as transferências que este está a fazer.

FS_Node: Cada Node tem duas threads a funcionar. Uma delas está a correr constantemente a ouvir por uma **socket UDP** e ao receber um pedido de outro Node chama uma das seguintes funções *envia_fileCmpl* caso o Node tenha o ficheiro completo na sua pasta ou então chama a função *envia_fileIncl* caso não tenha o ficheiro completo e apenas tenha alguns blocos. A segunda thread é a utilizada para comunicar com o Tracker via **TCP**. Quando é efetuado o pedido **get** e após se receber a resposta do Tracker é chamada a função *transf_file* para pedir os blocos. Nessa função é realizado o método de escalonamento escolhido para depois se pedir os blocos da forma mais eficiente aos Nodes. O algoritmo escolhido para escolher os blocos que se vai pedir aos Nodos foi o **Weigthed Round Robin** que iremos elaborar mais à frente como é que ele é aplicado.
Após terem sido selecionados os blocos que se vão pedir a cada Nodo é criada uma thread por Nodo que irá chamar a função *pedir_file* para cada um. Esta começa por calcular um intervalo de tempo com uma timestamp de quando envia uma mensagem *ping* e outra quando recebe a mensagem *pong*. No caso de não receber o *pong* entre o timeout inicialmente definido de 10 segundos é reenviado uma mensagem *ping* até um máximo de 3 pedidos. Após este timeout definido é então enviada a mensagem *request_block*, que funciona de forma semelhante à mensagem *ping*, sendo que no caso de passar o timeout é reenviado um pedido pelo bloco até um máximo de 3 tentativas. Caso no final das 3 tentativas o bloco não tenha sido recebido é então armazenado na lista *blocos_em_falta* o número do bloco que não se recebeu. Caso o bloco tenha chegado é então calculado um **checksum** dos **1198 bytes** para depois se comparar com os **2 bytes** da mensagem *send_block* que contêm o checksum. No caso de os valores não serem iguais é então armazenado na lista *blocos_em_falta* o bloco que não se recebeu direito. No caso de ter corrido tudo com sucesso é guardado no dicionário *blocos_recebidos* com a função *guarda_bloco_recebido*. Quando o Nodo que está a transferir o ficheiro tiver todos os blocos é chamada a função *escreve_file* que com o acesso ao dicionário *blocos_recebidos* irá escrever o ficheiro na pasta do Nodo e remove a informação desse ficheiro do dicionário.

FS_Tracker: Ao ser iniciado um Tracker num servidor é recolhido o seu **Ip** e é criada uma **socket TCP** com a **porta 9090** e o **Ip** anteriormente recolhido. Este socket

estará constantemente a ouvir pedidos de conexão de um *FS_ Node* sendo que quando recebe este pedido é criada uma thread que irá correr a função *handle_ node* para comunicar diretamente com cada Node até que um Node se desconecte: ao desligar-se diretamente ou ao enviar uma mensagem **quit**. Esta mensagem não foi mencionada nos protocolos pois não é necessária ao funcionamento do programa visto que não precisa de ser utilizada para desconectar o Node do Tracker como anteriormente mencionado.

Após se conectar um Node ao Tracker com a mensagem *files*, é então realizado o processo de armazenamento da informação sobre os ficheiros que o Node possui. Sendo então atualizado o dicionário *ficheiroDoNodo* com a nova informação. Este dicionário é composto por uma **chave** no formato de **String** que é composta pelo nome do ficheiro e a ele está vinculada a informação necessária de saber num **array** de três elementos: o primeiro elemento é o **número total de blocos** que esse ficheiro tem, o segundo elemento é uma **lista que contêm todos os Nodos** que possuem o ficheiro completo na sua pasta partilhada por fim o último elemento desse array é uma **lista de tuples** para os Nodos que não possuem o ficheiro completo mas têm pelo menos um bloco do mesmo, estes **tuples contêm um Nodo e os blocos que ele possui do ficheiro**. Relativamente aos Nodos, estes são armazenados em formato de **tuples**, sendo que o primeiro elemento é o nome do Node e o segundo é o seu **peso**. Este é a representação da sua qualidade a enviar ficheiros sendo que inicialmente todos os Nodos começam com um **peso** igual a 0 e ele vai sendo atualizado para cima ou para baixo consoante a taxa de sucesso do Node a enviar blocos. Existem diversas funções auxiliares que têm acesso a este dicionário sendo que as mais usadas são a *guarda_ localizacao* utilizada quando um Node se conecta, a *update_ info_ file* que atualiza a informação de um Nodo que está a transferir blocos e o peso do Node a quem ele está a pedir os blocos, a *procura_ file* que é a função utilizada para procurar um file que se pretende transferir devolvendo assim o conteúdo da mensagem *get_ response* e por fim, a função *remover_ info_ node* utilizada sempre que um Node se desconecta do Tracker.

O *FS_ Tracker* tem também acesso a uma lista *memoriaLogin* que é utilizada para recordar os Nodes que se desconectam do Tracker com um peso superior a 0. Assim quando um *FS_ Node* se conecta no caso de estar presente na lista anteriormente mencionada o seu **peso** passa a ser o recordado, caso contrário é 0, sendo então removido da lista da *memoriaLogin* enquanto está conectado.

DNS: Para o desenvolvimento de um sistema DNS utilizamos o servidor bind9, para facilitar a resolução de nomes de hosts e endereços IP. Desta forma no arquivo de configuração principal **named.conf.local** foram criados duas zones com o nome **"CC2023"** e **"10.in-addr.arpa"**. A primeira, "CC2023", foi configurada como uma zona mestre, indicando que este servidor é a autoridade primária para essa zona. A segunda, "10.in-addr.arpa", foi configurada para lidar com consultas de reverso.

4.2 Parâmetros

O algoritmo que escolhemos para escalonar os blocos a pedir pelos Nodos, tal como mencionado anteriormente, foi o **Weighted Round Robin** que utiliza os pesos para priorizar a que Nodos vai pedir mais blocos.

Na função *transf_file* começamos por verificar quantos Nodos individuais existem. No caso de só existir um Nodo com o ficheiro não é realizado nenhum processo de escolha de nodos pois não é necessário, apenas são chamadas as funções que pegam na informação que o *FS_Tracker* envia com a mensagem *get_response* e as organiza para que a função *pedir_file* possa pedir os blocos ao Nodo. Caso a função *verifica_existe_prioridade* (que compara todos os pesos dos Nodos e verifica se existe algum com mais peso que os outros) retorne 0, significa que todos os Nodos que contêm o ficheiro têm o mesmo peso sendo então repartido de forma igual pelos Nodos os blocos que se vai pedir a cada um. Caso seja retornado o valor de 1 é então dada prioridade aos Nodos que têm maior pontuação para se pedir mais blocos. A função *escolhe_nodos* utiliza o algoritmo selecionado (**WRR**) para escolher o Nodo que tem maior peso e possui o bloco até ao limite de blocos que cada Nodo pode enviar para não haver um Nodo sobrecarregado com pedidos. Esta função irá criar as listas necessárias para o caso de haver mais blocos do que o limite permite dividir pelos Nodos. Antes de executar estas funções é chamada a função *ordena_por_nodos* que **dá prioridade aos blocos que estão em menos Nodos**, sendo que estes blocos serão os primeiros a ser pedidos e os blocos que estão em mais blocos serão os últimos a ser requisitados.

4.3 Bibliotecas de funções

Nós dividimos o projeto em 5 classes diferentes que interagem entre si. Temos 2 classes que estão relacionadas com o *FS_Tracker*: uma delas é a classe *FS_Tracker* que têm tudo o que é necessário para o Tracker comunicar com o Nodo e a outra classe é a *Struct_FileNodos* que é a classe que trata das estruturas de informações a que o Tracker têm acesso (*ficheirosDoNodo* e *memoriaLogin*). As restantes 3 classes são relacionadas ao *FS_Node* sendo elas a *FS_Node* que contêm as funções necessárias para o Nodo interagir com o Tracker, possui o dicionário *blocos_recebidos* e contêm a função que mantêm a socket **UDP** ativa. A classe *Metodo_Transf* é responsável pelas funções que tratam da transferência dos blocos e onde está presente a lista *blocos_em_falta*. Por fim a classe *Metodo_SelecNodes* contém todas as funções utilizadas para transformar o conteúdo da mensagem *get_response* e *updlst_response* e transformam na informação utilizada para pedir os blocos, sendo que também está aqui presente as funções que aplicam o algoritmo **Weighted Roun Robin** para escolher os Nodos a quem se vai pedir os blocos.

5 Testes e resultados

De forma a demonstrar o programa em funcionamento foram realizados diversos testes, começamos por demostrar a ativação do sistema DNS.

Figura 14. Comando de ativação do DNS

De seguida vemos as ações que ocorrem assim que um *FS_ Node* se conecta ao *FS_ Tracker*, como podemos observar o Node começa com um peso igual a 0.

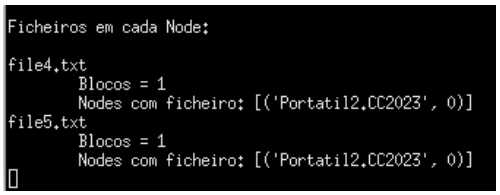

Figura 15. Tracker com as informações do Node

De seguida temos uma transferência de um ficheiro por um PC com má conexão.

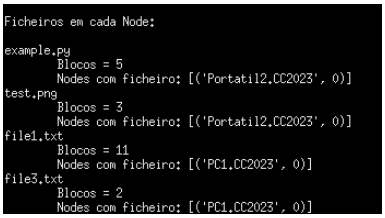

Figura 16. Tracker com as informações antes da transferência

Figura 17. Requesição e chegada dos blocos no PC1

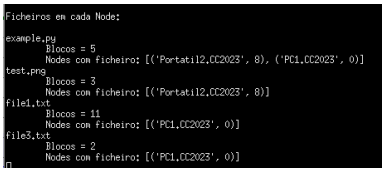

Figura 18. Tracker com as informações durante a transferência

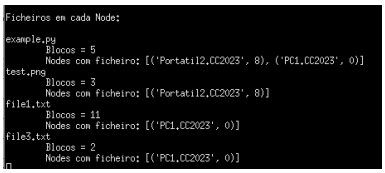

Figura 19. Tracker com as informações depois da transferência

Por fim demonstramos a transferência que ocorre em paralelo a pedir o mesmo ficheiro a dois Nodos diferentes que contêm o mesmo ficheiro.

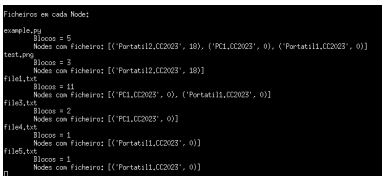

Figura 20. Tracker com as informações relativas aos Nodos antes da transferência

Como é possível observar vemos que são requisitados mais blocos ao Portátil2 visto que este têm melhor pontuação que o PC1 no começo da transferência. Neste exemplo conseguimos aferir também que o Portátil um não recebe a reply *pong* sendo que volta a enviar um *ping*.

Figura 21. Requisições e receção dos blocos em Paralelo

6 Conclusões e trabalho futuro

Este trabalho foi dividido em três partes gerais como recomendado no enunciado. Na primeira parte, concentramo-nos no design e implementação do protocolo FS Track Protocol sobre TCP.

Na segunda fase, dirigimos a nossa atenção para o protocolo FS Transfer Protocol, desenvolvendo uma solução baseada em UDP.

Na última fase, introduzimos um serviço de resolução de nomes "DNS", onde os *FS_ Node* e o *FS_ Tracker* identifican-se através de nomes.

Em última análise, este trabalho permitiu-nos aprofundar na criação de serviços avançados de partilha de ficheiros em ambientes P2P. Ao consolidar diversos conceitos teóricos e aplicá-los de forma prática, obtivemos abilidades valiosas que serão importantes para os nossos futuros estudos e trabalhos na área de Comunicações por Computador.