Relatório TP2

Pedro Veloso[1 [a89557]], Carlos Preto[1 [a89587]], and Joan Rodriguez[1 [a89980]]

Universidade do Minho, Braga, Portugal
http://www.uminho.pt

Abstract. Neste relatório serão abordados os tópicos referentes ao desenvolvimento e implementação do protocolo aplicacional desenvolvido.

Keywords: TCP · HTTP · UDP.

1 Introdução

Neste trabalho pretende-se desenvolver um protocolo aplicacional capaz de funcionar como intermediário no pedido de um ficheiro.
Com a criação do gateway pretende-se que o download de um arquivo seja dividido entre os vários servidores conectados, diminuindo assim a carga de trabalho de cada servidor. Contudo, uma vez que os arquivos se encontram nos servidores, estando assim em diferentes máquinas, é necessário realizar vários pedidos, utilizando para tal efeito o UDP, uma vez que, não sendo orientado à conexão é mais rápido na transferência de arquivos.
Para a troca de mensagens entre os servidores e o HttpGw desenvolveu-se um formato para as mensagens protocolares, que contém sete campos sendo eles referentes ao tipo da mensagem transportada, ao nome do arquivo que se pretende descarregar, ao conteúdo do arquivo e como as mensagens são transportadas via UDP existe também um campo que indica se os dados recebidos estão completos, ou se é necessário receber ainda mais mensagens.
Ao longo deste relatório pretende-se, explicar o funcionamento da aplicação desenvolvida, bem como especificar em detalhe todos os campos existentes nas mensagens protocolares, para além da sua importância.

2 Arquitetura da solução

Com a realização deste trabalho pretende-se implementar um *gateway* de aplicação, designado por *HttpGw*, capaz de receber múltiplos pedidos em simultâneo e recorrendo a um conjunto de FastFileSrv's disponibilizar os ficheiros pedidos.
Para a implementação do *gateway*, uma vez que é necessário atender múltiplos pedidos em simultâneo, considerou-se a criação de 2 *threads* principais, um responsável por receber os pedidos do cliente, outro responsável por receber e processar as mensagens oriundas dos FastFileSrv's (início de conexão, receber os dados dos ficheiros pedidos e finalização de conexão). Contudo, o *HttpGw*, não se limita a estes dois *threads*, criando mais *threads* responsáveis, por exemplo, por pedir ou receber *chuncks* de dados/metadados. Assim o *gateway* terá em memória os metadados (nome e tamanho) referentes aos ficheiros que cada servidor possui.
Na figura seguinte exemplifica-se o funcionamento da aplicação, para um pedido de inicio de conexão e para um pedido de um ficheiro.

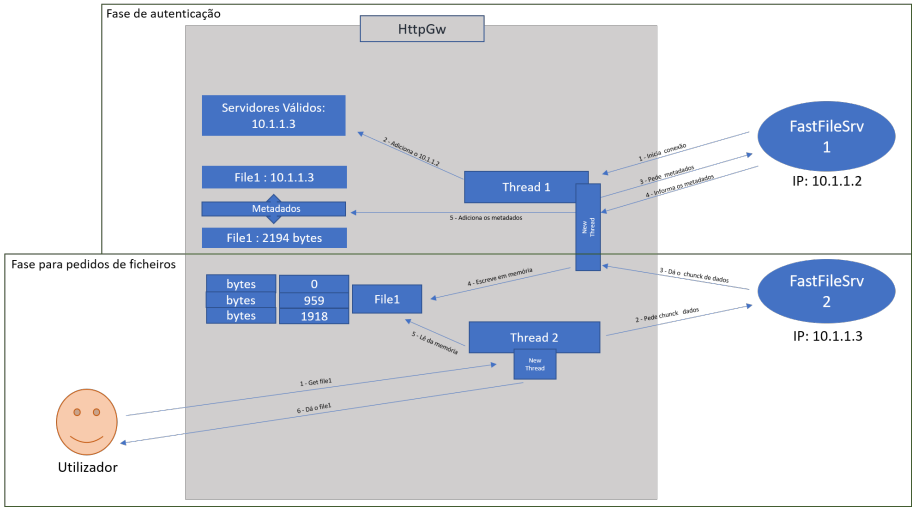


Fig. 1. Esquematização do protocolo aplicacional

3 Especificação do protocolo

3.1 Sintaxe

Para a comunicação entre o *HttpGw* e o *FastFileSrv* desenvolveu-se um formato para as mensagens trocadas entre ambos. Para a definição destas mensagens definiu-se como tamanho máximo 1024, pois este valor é o valor de referência para evitar a fragmentação dos pacotes.
O formato desenvolvido corresponde ao apresentado na Figura 2 e apresenta 7 campos distintos – tipo, offset, length_nome, nome_ficheiro, length, dados e flag.

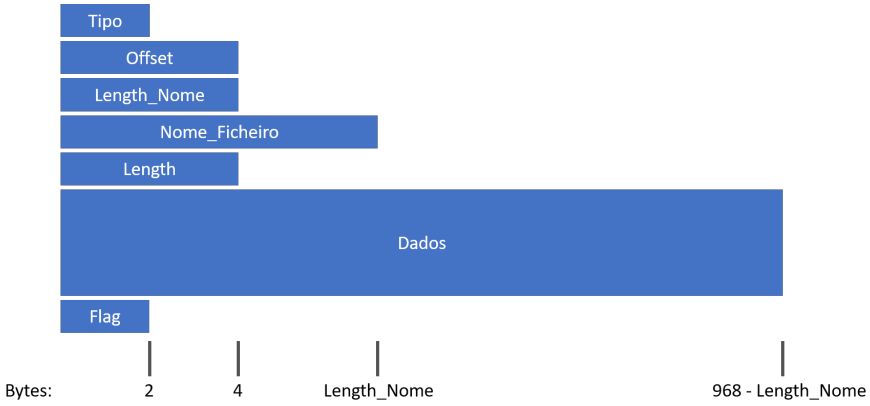


Fig. 2. Esquematização do protocolo desenvolvido

3.2 Semântica

Analisando a figura 1 nota-se a existência de 7 campos na mensagem, sendo que cada campo desempenha um papel fundamental no protocolo desenvolvido.

Tipo: O primeiro campo identificado nas mensagens protocolares, é referente ao tipo da mensagem. Uma vez que existem 4 tipos distintos de mensagens protocolares – início de conexão, fim da conexão, envio de metadados e envio de dados – reserva-se o uso de 2 bytes para identificar os casos distintos. Assim, usa-se a seguinte notação:

- Caractere I: Refere uma mensagem enviada para iniciar uma conexão entre o HttpGw e um FastFileSrv;
- Caractere G: Refere uma mensagem enviada para pedir/enviar metadados de um servidor;

- Caractere T: Refere uma mensagem enviada para pedir/enviar dados de um ficheiro;
- Caractere F: Refere uma mensagem enviada para finalizar a conexão entre o HttpGw e o FastFileSrv.

Como referido utiliza-se um caractere para identificar o tipo da mensagem, o que implica o uso de 16 bits (2 bytes) do total da mensagem protocolar. Contudo, como apenas existem 4 tipos distintos seria possível distingui-los usando apenas 3 bits, no entanto implicaria tratar a mensagem bit a bit, dificultando a sua implementação.

Offset: O campo *offset* é utilizado para identificar, tal como o nome indica, a distância a que o *chunk* de dados transportado se encontra do início dos dados totais. Assim é possível diferenciar a utilização do campo *offset* nos diferentes casos:

- O FastFileSrv recebe uma mensagem para o pedido de dados/metadados. Neste caso deve ser enviado os dados, a começar no *offset* e a terminar no tamanho máximo reservado para o campo dados.
- O HttpGw recebe uma mensagem de transferência de dados/metadados. Neste caso recebe-se uma porção de dados, onde o seu início nos dados totais corresponde ao *offset*.

Sendo o *offset* um inteiro é utilizado 4 bytes do tamanho total da mensagem protocolar para este campo.

Length_Nome: O campo denominado como "Length_Nome" é utilizado para determinar o tamanho, em bytes, do comprimento de um outro campo – campo referente ao nome do ficheiro a que os dados transportados são referente.
Este campo existe porque o tamanho do nome do ficheiro é variável. Uma vez que o campo se relaciona com o nome do ficheiro transportado, no caso de não existir um ficheiro o valor deste campo deve ser 0.
Como o tamanho é dado por um inteiro, reserva-se para este campo 4 bytes.

Nome_Ficheiro: Este campo deve ser utilizado apenas no tipo de mensagem referente ao transporte de dados, uma vez que é apenas nesses casos que existem dados relacionados aos ficheiros.
A existência deste campo é fundamental para o HttpGw conhecer o ficheiro a que os dados recebidos se associam. Também é importante para o FastFileSrv saber de que ficheiro deve ler os dados. Uma vez que o tamanho do nome do ficheiro é variável, o tamanho, em bytes, deste campo é dado pelo campo "Lenght_Nome".

Length: Uma vez mais, utiliza-se um campo para definir o tamanho de um outro campo. Assim o campo "Length" é utilizado para especificar o tamanho, em bytes, dos dados transportados. Caso não sejam transportados nenhuns dados, este campo deve tomar como valor o valor 0.
Para este campo reserva-se mais 4 bytes da mensagem protocolar.

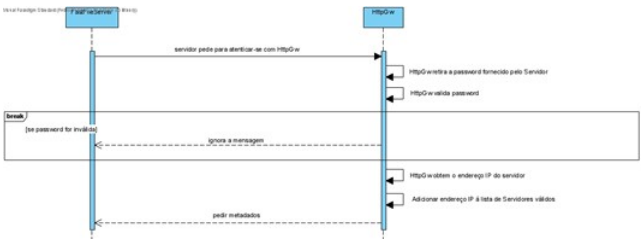
Dados: Este campo é usado para transportar dados referentes aos dados dos ficheiros ou aos metadados dos servidores. No início da conexão é também neste campo que deve ser transportado a password para autenticação com o HttpGw. Este campo também tem tamanho variável, que vai especificado no campo "Length".

Flag: Este campo corresponde a uma *flag* que indica se a mensagem recebida é a última, ou se ainda existe mais mensagem posteriores à recebida, que devem ser consideradas. Para este campo, de modo a simplificar o desenvolvimento do protocolo, utiliza-se um caractere que indica se a *flag* se encontra a *true* ou não. Contudo, caso se optasse por tratar os dados bit a bit seria possível implementar este campo utilizando apenas um bit.

3.3 Comportamento

Durante o funcionamento do HttpGw serão trocadas muitas e diferentes mensagens, com os diferentes FastFileSrv's, apresentando diferentes comportamentos em algumas delas. Contudo, é possível generalizar o comportamento de todas as mensagens trocadas nos seguintes casos:

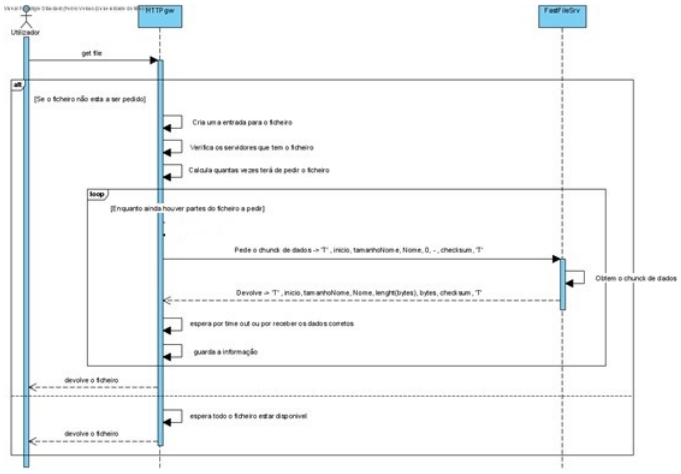
Mensagens de início de conexão: Sempre que um FastFileSrv pretende conectar-se ao HttpGw deverá enviar-lhe uma mensagem de modo que o HttpGw o adicione à lista de servidores conhecidos e admitidos.
A mensagem enviada deverá ser do tipo I, indicando o início de conexão e deverá conter no campo dados a password válida e conhecida pelo HttpGw. O offset e o tamanho do nome nesta mensagem irão a 0, uma vez que não são transportados dados relativos a ficheiros. A *flag* deverá ir a verdadeiro uma vez que a mensagem é única.
Ao receber um pedido com as características referidas, o HttpGw deverá adicionar o endereço do servidor à lista de endereços válidos e iniciar a troca de metadados.



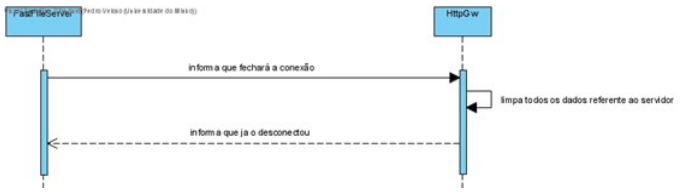
Mensagens para troca de metadados: Após o início de conexão o servidor estará à espera de um pedido dos metadados (é este pedido que dá a confirmação

ao servidor que o início de conexão foi recebido e só depois de receber um pedido deste efeito é que o servidor pode considerar que a conexão foi iniciada) preparando toda a informação sobre os ficheiros disponíveis, bem como o seu tamanho.
Ao receber um pedido de metadados o servidor deverá repartir esses dados em *chuncks* e enviar uma mensagem ao HttpGw do tipo G anexada com os dados. Como a informação pode ser enviada em várias mensagens o servidor deverá colocar a flag a falso em todas as mensagens com exceção da última, para além de preencher corretamente o campo do *offset*.
Já o HttpGw ao receber uma mensagem de troca de metadados deverá ir acumulando a informação recebida. Após o armazenamento dos dados recebidos o HttpGw deverá, em caso de a mensagem recebida não ser a última, pedir mais metadados, com o *offset* desejado.

Mensagens para troca de dados de ficheiros: Quando o HttpGw recebe um pedido de um ficheiro, deve verificar quais servidores contém aquele arquivo e em quantas mensagens aquele ficheiro vai ser transportado. Após isto deve pedir um chunck de dados a cada servidor, em estilo *round-robin*.
Ao receber deve guardar a informação associando-a ao seu *ofsset*, o que permitirá juntar todas as partes ordenadas e construir o ficheiro completo sem erros.



Mensagem para finalizar conexão: Quando um servidor pretender finalizar uma conexão deverá enviar uma mensagem com o tipo F e com a *flag* a verdadeiro. Para os campos referentes ao *offset*, ao tamanho do nome e dos dados deverá ser associado a cada um deles o valor 0, uma vez que não serão transportados quaisquer dados.
Ao receber uma mensagem deste tipo o HttpGw deverá limpar toda a informação sobre o emissor.



4 Implementação

Para implementar o protocolo aplicacional referido e explicado ao longo deste relatório, recorreu-se à linguagem JAVA.
Uma vez que o funcionamento geral da aplicação desenvolvida já foi abordada de uma forma geral nos capítulos anteriores, nesta secção apenas se abordará os recursos utilizados para registar os dados dos ficheiros recebidos, bem como a estratégia desenvolvida para aguardar a receção de todo o ficheiro.

Registar os dados: Para registar os vários *chuncks* de dados que o HttpGw recebe, criou-se um *map* que contém como chave o nome do ficheiro. Associado ao nome do ficheiro está um outro *map* que desta vez associa o *offset* dos dados ao verdadeiro conteúdo do ficheiro (*byte[]*). Recorrendo a esta estrutura de dados, é possível registar os *chuncks* de dados, associado-os tanto ao ficheiro, como ao *offset*. Posteriormente, a montagem do ficheiro também se torna simples, uma vez que basta obter o *map* que contém os *offset* e os *bytes* e ordenar pela chave, i.e. ordenar pelo *offset*.

Aguardar pelo receção de todos os dados: Um processo importante na aplicação desenvolvida é o pedido dos dados aos FastFileSrv's, contudo sendo este processo desenvolvido com o suporte do UDP, foi necessário garantir que os pedidos dos dados atingia os servidores. Assim criou-se um responsável por pedir os dados ao servidor até obter uma resposta. Contudo de modo a evitar esperas ativas utilizou-se a interface "Condition" definida em JAVA. Assim, é possível concluir que o ficheiro está totalmente em posse do HttpGw, quando todos os *Thread's*, responsáveis por pedir os dados, terminarem a sua execução.

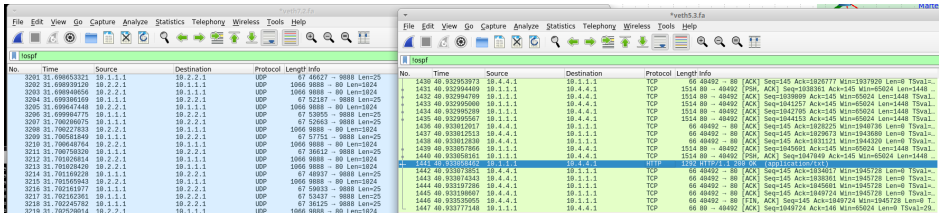
5 Testes e resultados

Após o desenvolvimento da aplicação realizou-se vários testes que permitiam garantir a viabilidade da aplicação. Assim neste capítulo pretende-se demonstrar alguns dos resultados obtidos.

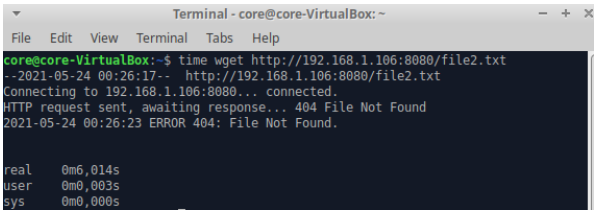
Teste 1: No primeiro teste, realizou-se pedidos de um ficheiros com tamanhos variados de modo a perceber se o ficheiro era obtido sem erros.

```
core@core-VirtualBox:~$ time wget http://192.168.1.64:8080/video.mp4
--2021-05-23 17:53:13--  http://192.168.1.64:8080/video.mp4
Connecting to 192.168.1.64:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11664390 (11M) [application/mp4]
Saving to: 'video.mp4.1'

video.mp4.1         100%[===================>]  11,12M  38,0MB/s    in 0,3s

2021-05-23 17:53:18 (38,0 MB/s) - 'video.mp4.1' saved [11664390/11664390]
```

Teste 2: Num segundo teste, verificou-se, recorrendo ao *wireshark*, se a ligação entre o cliente e o HttpGw e entre o HttpGw e os FastFileSrv's estavam a ser realizadas de forma correta.



Teste 3: Por fim realizou-se um teste de modo a perceber o comportamento obtido caso um FastFileSrv encerre a conexão repentinamente, sem se desconectar do HttpGw.



6 Conclusões e trabalho futuro

A realização deste trabalho permitiu desenvolver um servidor capaz de receber pedidos HTTP, realizar o seu *parsing* e recorrendo a um outro conjunto de servidores devolver o ficheiro ao cliente.
Apesar do desenvolvimento do trabalho permitir obter todos as funcionalidades pedidas, existem alguns aspetos que poderão ser melhorados no futuro. Assim, uma vez que os FastFileSrv são máquinas independentes, pode ocorrer a situação em que vários servidores têm ficheiros com conteúdo diferente, mas com o mesmo nome, o que implicaria um problema para o HttpGw pois consideraria que o ficheiro era o mesmo.
Para além deste problema identificado, existem outros relacionados com a segurança que o HttpGw fornece, com a sua utilização, pois a estratégia de criar um *Thread* para cada pedido, torna-o vulnerável, por exemplo, a ataques de *denial of service*.