Trabalho Prático 1 - Protocolos da Camada de Transporte

Comunicações por Computador

PL1 - Grupo 03

Gabriela Santos Ferreira da Cunha - a97393

Miguel de Sousa Braga - a97698

Nuno Guilherme Cruz Varela - a96455

Conteúdo

1 Parte B : Questões

1.1 Exercício 1

De que forma as perdas e duplicações de pacotes afetaram o desempenho das aplicações? Que camada lidou com esses problemas: transporte ou aplicação? Responda com base nas experiências feitas e nos resultados observados.

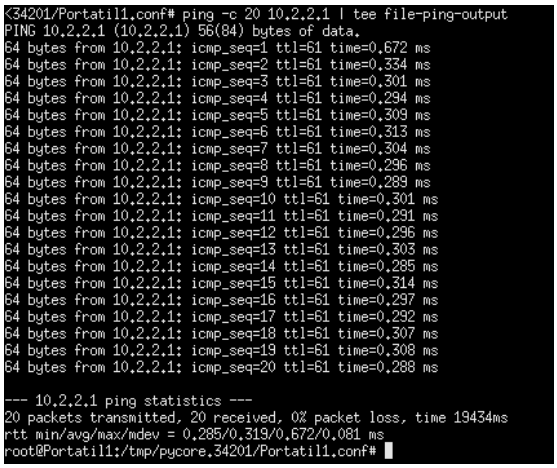

Figura 1: Ping do Portatil1 para o Servidor1

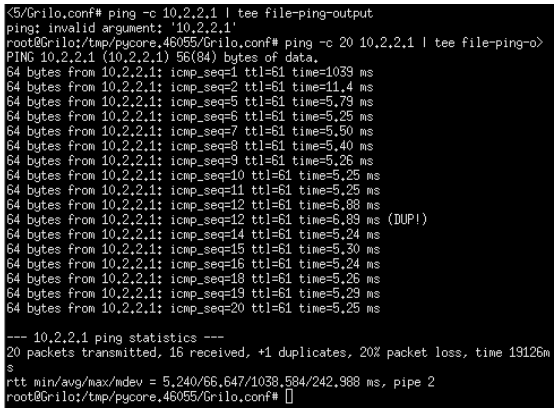

Figura 2: Ping do Grilo para o Servidor1

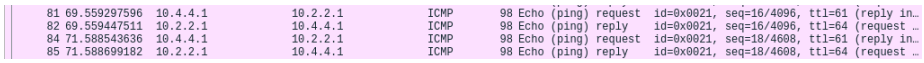

Figura 3: Perda de um pacote enviado pelo Grilo para o Servidor1

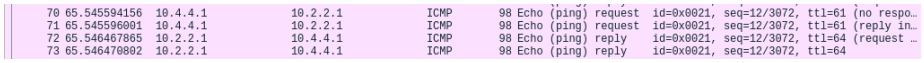

Figura 4: Duplicação de um pacote enviado pelo Grilo para o Servidor1

Como podemos observar a partir da figura 1, foram enviados e recebidos 20 pacotes do Portatil1 para o Servidor1, ou seja, não houve qualquer tipo de perda ou duplicação. Já no ping efetuado do Grilo para o Servidor1 podemos verificar perdas, através do número de sequência, e duplicações de pacotes.

Na figura 3, o número de sequência saltou de 16 para 18, o que nos permite concluir que houve a perda do pacote com *icmp_seq* 17. Na figura 4, podemos ver que foram enviados dois *requests* com *icmp_seq* 12, por isso houve aqui uma duplicação do pacote. Com isto, conseguimos verificar que o ping não tem qualquer controlo sobre este tipo de situações, uma vez que o protocolo utilizado é o "ìcmp" que é um protocolo da camada de rede e estas perdas e duplicações são, por isso, visíveis para quem utiliza a aplicação.

Estas perdas e duplicações de pacotes afetam negativamente o desempenho das aplicações na medida em que perder pacotes não é algo positivo quando queremos ter serviços de alta precisão, assim como duplicar pacotes é algo ineficiente, dado que o servidor está a fazer um esforço extra para responder ao mesmo pedido.

Através das experiências de transferência de ficheiros realizadas, conseguimos concluir que o protocolo de transporte TCP é um protocolo que assegura ligações fiáveis, visto que a própria camada de transporte lida com as perdas e duplicações de pacotes. Para evitar estes problemas, este protocolo garante uma sequenciação dos pacotes, através dos campos do cabeçalho "sequence number" e "acknowledgment number" e envia pacotes de confirmação - *Acknowledgment*. Contudo, todo este processo envolve uma maior complexidade, o que pode originar uma certa latência e atraso nas aplicações.

Por outro lado, o protocolo de transporte UDP não trata deste tipo de problemas, pelo que estas perdas e duplicações são tratadas pela camada de aplicação. Com isto, conseguimos ter ligações mais rápidas, porque não temos tantos pacotes de controlo a serem trocados.

1.2 Exercício 2

Obtenha a partir do Wireshark, ou desenhe manualmente, um diagrama temporal para a transferência do ficheiro file1 por FTP realizada em A.3. Foque-se apenas na transferência de dados [ftp-data] e não na conexão de controlo (o FTP usa mais que uma conexão em simultâneo). Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão. Identifique também os tipos de segmentos trocados e os números de sequência usados tanto nos dados como nas confirmações.

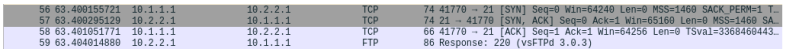

Figura 5: Conexão FTP efetuada pelo Portatil1 ao Servidor1

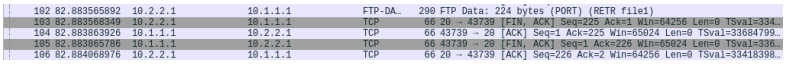

Figura 6: Transferência de dados do Servidor1 para o Portatil1

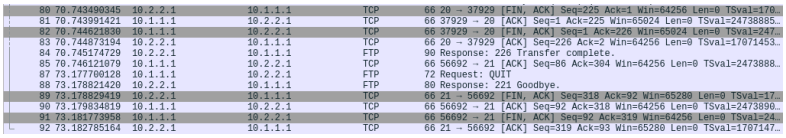

Figura 7: Fim da conexão do Portatil1 com o Servidor1

O FTP (File Transfer Protocol) é um protocolo de transferência de ficheiros que trabalha sobre o protocolo de transporte TCP.

Neste caso, utilizamos este protocolo para analisar a transferência do ficheiro "file1" do Servidor1 para o Portatil1. Neste exemplo, são usadas 2 conexões simultâneas para efetuar esta ligação.

A primeira estabelece-se após o uso do comando "ftp" na *bash* do Portatil1. Este desencadeia uma sequência de segmentos TCP e FTP (também usa TCP) trocados entre as duas máquinas. Primeiro, tal como podemos ver na Figura 5, é estabelecida a ligação com recurso ao *3-way handshake*. Em seguida, é realizada a autenticação e troca de comandos como "status" e "dir".

A segunda conexão é criada com o propósito de transferir o ficheiro em questão - "ftp_data" - assim que é executado o comando "get file1" no Portatil1.

Por fim, ambas as ligações são fechadas (Figura 7).

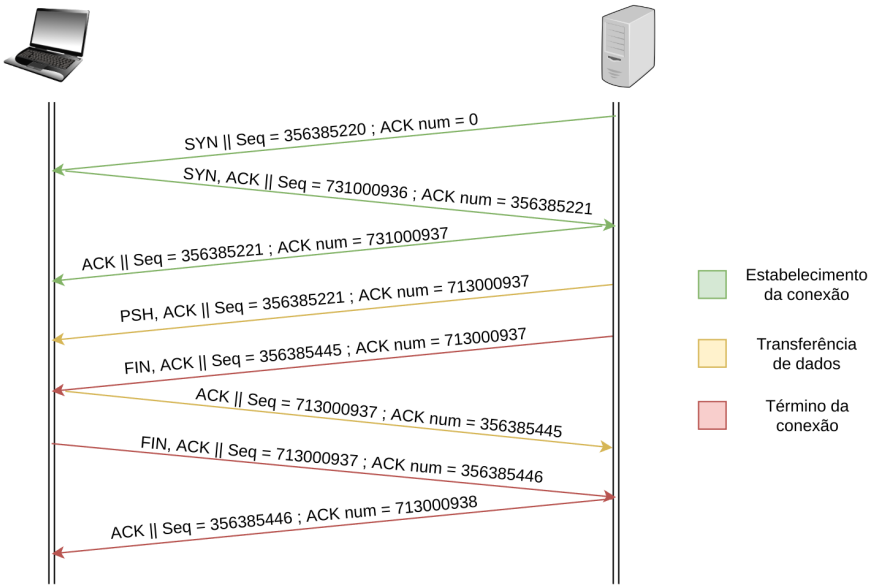

Figura 8: Diagrama temporal da transferência do ficheiro por FTP.

Analisando agora, em detalhe, as 3 fases da conexão para a transferência do ficheiro de dados, elaboramos o esquema que se encontra acima representado. Este pretende representar os segmentos trocados de forma *full-duplex* entre as duas máquinas.

A ligação estabelece-se com a troca de 3 segmentos (*3-way handshaking*): SYN, SYN-ACK e ACK. Este processo tem como objetivo a negociação dos termos para a conexão e a sincronização dos números de sequência, tanto para o servidor como para o portátil. Quem procura iniciar a conexão é o servidor, pois é este que pretende enviar o ficheiro para o cliente. Assim, envia um segmento sem dados com a flag SYN ativa e um determinado número de sequência "x"de natureza aleatória, indicando a numeração do primeiro *byte* de dados a ser enviado. O cliente Portatil1, ao receber esse segmento, já sabe que o próximo segmento a receber do servidor terá de ser o segmento "x+1". Como podemos ver no diagrama, este é o número que aparece no campo ACK num do segundo segmento enviado. Para além disso, o cliente envia ao servidor o seu próprio número de sequência inicial. Este é, por isso, um segmento do tipo SYN (sincronização) e ACK (*acknowledgment*), pois sincroniza e confirma a conexão com o servidor. Por fim, um último segmento ACK é enviado do Servidor1 para o Portatil1, indicando que este recebeu os termos do Portatil1 e que a transferência de dados pode ser iniciada.

A transferência de dados propriamente dita dá-se com a troca de apenas 2 segmentos: um PSH-ACK do Servidor1 para o Portatil1 e a respetiva confirmação ACK do Portatil1 para o Servidor1. Podemos afirmar que o ACK mencionado está, de facto, a confirmar o PSH do servidor, pois este indica que o número de sequência do próximo segmento a receber é o 356385445, valor que está presente no segmento FIN-ACK proveniente do servidor.

O segmento FIN-ACK enviado pelo Servidor1 estabelece o início do fim da conexão. Com o envio dessa mensagem, o Servidor1 indica ao Portatil1 que não pretende enviar mais dados. O cliente responde com um outro FIN-ACK, indicando também que, da sua parte, não tem mais dados para enviar e testemunhando a receção do FIN-ACK proveniente do servidor. Por fim, o Servidor1 envia um último segmento ACK, confirmando a receção do FIN-ACK do Portatil1 e terminando, de forma efetiva, a conexão.

1.3 Exercício 3

Obtenha a partir do Wireshark, ou desenhe manualmente, um diagrama temporal para a transferência do ficheiro file1 por TFTP realizada em A.4. Identifique, se aplicável, as fases de início de conexão, transferência de dados e fim de conexão. Identifique também os tipos de segmentos trocados e os números de sequência usados tanto nos dados como nas confirmações.

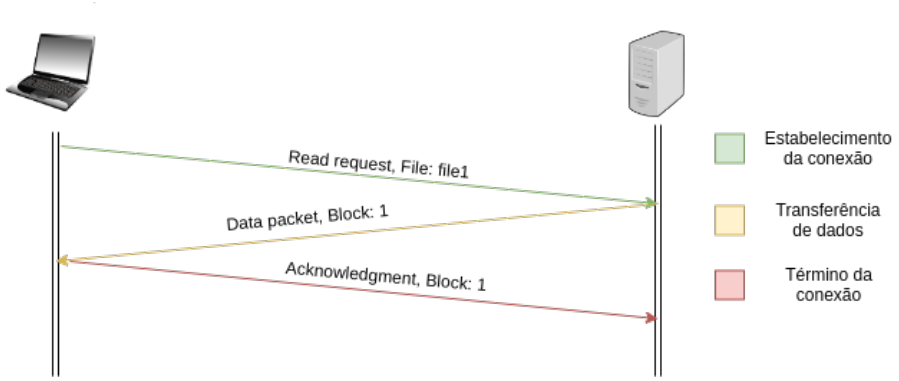

Figura 9: Diagrama temporal da transferência do ficheiro por TFTP.

O TFTP (Trivial File Transfer Protocol) é um protocolo de transferência de ficheiros que trabalha sobre o protocolo de transporte UDP.

Como podemos observar pela figura 9, o Portatil1 envia um pedido de leitura para o Servidor1. Em seguida, o Servidor1 envia o pacote de dados e o Portatil1 confirma a sua receção através do envio de um pacote TFTP ACK.

Em comparação com o FTP, neste protocolo o número de pacotes trocados é bastante mais baixo e não existem *sequence numbers*, visto que o TFTP trabalha sobre UDP e, por isso, utiliza overheads menores (cabeçalhos de 8 *bytes*), o que torna a transferência mais eficiente e menos fiável.

1.4 Exercício 4

Compare sucintamente as quatro aplicações de transferência de ficheiros que usou, tendo em consideração os seguintes aspetos: (i) identificação da camada de transporte; (ii) eficiência; (iii) complexidade; (iv) segurança.

SFTP O primeiro protocolo (e aplicação) usado para tranferência de ficheiros foi o SSH File Transfer Protocol (SFTP). Tal como o nome indica, este protocolo usa o protocolo SSH (v2) para transferir os dados de forma segura através de um determinado link. Os dados são encriptados e enviados para o destino. O protocolo SFTP corre em cima do protocolo TCP, garantindo assim uma ligação de dados fiável. Com o envio de *Acknowledgments*, o servidor permite saber se os dados foram recebidos com sucesso a quem está a enviar informação. Este é um protocolo com um nível de complexidade superior, uma vez que recorre a mecanismos de autenticação e encriptação sofisticados, recorrendo a vários tipos diferentes de mensagens, tal como podemos ver na figura 11. Tal como iremos mostrar no exercício 5, esta forma de transferir ficheiros é relativamente pouco eficiente, comparada por exemplo com a aplicação TFTP, visto que a percentagem de overhead no total de bytes transferidos é considerável.

```
root@Portatil1:/tmp/pycore.38059/Portatil1.conf# rm /root/.ssh/known_hosts
root@Portatil1:/tmp/pycore.38059/Portatil1.conf# sftp core@10.2.2.1
The authenticity of host '10.2.2.1 (10.2.2.1)' can't be established.
RSA key fingerprint is SHA256:ZuHjVnq/p6X1LsNIM9k+2CMu6eipi42/int4wz7kwJY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.2.2.1' (RSA) to the list of known hosts.
core@10.2.2.1's password:
Connected to 10.2.2.1.
sftp> pwd
Remote working directory: /home/core
sftp> cd /srv/ftp
sftp> dir
file1  file2
sftp> get file1
Fetching /srv/ftp/file1 to file1
/srv/ftp/file1                                100%  224    45.1KB/s   00:00
sftp> quit
root@Portatil1:/tmp/pycore.38059/Portatil1.conf#
```

Figura 10: Transferência SFTP Grilo

```
47 54.028582764  10.1.1.1    10.2.2.1    TCP      74 53032 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
48 54.028737168  10.2.2.1    10.1.1.1    TCP      74 22 → 53032 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
49 54.028951105  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1785126123…
50 54.033797981  10.1.1.1    10.2.2.1    SSHv2   107 Client: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5)
51 54.034768127  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=112615832…
52 54.037650666  10.2.2.1    10.1.1.1    SSHv2   107 Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5)
53 54.037814613  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=17851261…
54 54.037936627  10.1.1.1    10.2.2.1    TCP    1514 53032 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=1448 TSval=17851…
55 54.037937126  10.1.1.1    10.2.2.1    SSHv2   130 Client: Key Exchange Init
56 54.038126599  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=42 Ack=1490 Win=64128 Len=0 TSval=112615…
57 54.038128054  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=42 Ack=1554 Win=64128 Len=0 TSval=112615…
58 54.039217090  10.2.2.1    10.1.1.1    SSHv2   1090 Server: Key Exchange Init
59 54.039605609  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1554 Ack=1066 Win=64128 Len=0 TSval=1785…
60 54.040970280  10.1.1.1    10.2.2.1    SSHv2   114 Client: Diffie-Hellman Key Exchange Init
61 54.041546286  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=1066 Ack=1602 Win=64128 Len=0 TSval=1126…
62 54.048879138  10.2.2.1    10.1.1.1    SSHv2   1182 Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypte…
63 54.049316901  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1602 Ack=2182 Win=64128 Len=0 TSval=1785…
64 56.018989367  10.2.2.254  224.0.0.5   OSPF     78 Hello Packet
65 56.197716350  10.1.1.1    10.2.2.1    SSHv2    82 Client: New Keys
66 56.197838716  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=2182 Ack=1618 Win=64128 Len=0 TSval=1126…
67 56.197981825  10.1.1.1    10.2.2.1    SSHv2   110 Client: Encrypted packet (len=44)
68 56.198121572  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=2182 Ack=1662 Win=64128 Len=0 TSval=1126…
69 56.198122870  10.2.2.1    10.1.1.1    SSHv2   110 Server: Encrypted packet (len=44)
70 56.198256948  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1662 Ack=2226 Win=64128 Len=0 TSval=1785…
71 56.198283036  10.1.1.1    10.2.2.1    SSHv2   126 Client: Encrypted packet (len=60)
72 56.198992609  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=2226 Ack=1722 Win=64128 Len=0 TSval=1126…
73 56.205468934  10.2.2.1    10.1.1.1    SSHv2   118 Server: Encrypted packet (len=52)
74 56.246519134  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1722 Ack=2278 Win=64128 Len=0 TSval=1785…
75 58.020580289  10.2.2.254  224.0.0.5   OSPF     78 Hello Packet
76 58.148560243  10.1.1.1    10.2.2.1    SSHv2   150 Client: Encrypted packet (len=84)
77 58.156409484  10.2.2.1    10.1.1.1    SSHv2    94 Server: Encrypted packet (len=28)
78 58.157516038  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1806 Ack=2306 Win=64128 Len=0 TSval=1785…
79 58.157516597  10.1.1.1    10.2.2.1    SSHv2   178 Client: Encrypted packet (len=112)
80 58.198555334  10.2.2.1    10.1.1.1    TCP      66 22 → 53032 [ACK] Seq=2306 Ack=1918 Win=64128 Len=0 TSval=1126…
81 58.636126847  10.2.2.1    10.1.1.1    SSHv2   534 Server: Encrypted packet (len=468)
82 58.637081067  10.1.1.1    10.2.2.1    TCP      66 53032 → 22 [ACK] Seq=1918 Ack=2774 Win=64128 Len=0 TSval=1785…
```

Figura 11: Complexidade da transferência SFTP Portatil1

FTP O FTP (File Transfer Protocol) é um outro protocolo de tranferência de ficheiros. Neste caso, não é realizada a encriptação dos dados, pelo que este é menos seguro que o anterior. As *passwords* são enviadas em modo texto através da rede e qualquer um pode ter acesso. No entanto, a comunicação continua a ser fiável, uma vez que este protocolo também usa como protocolo da camada de transporte o protocolo TCP. Quanto à complexidade, através de uma visita rápida ao site "`https://www.rfc-editor.org/rfc/rfc959`", pudemos analisar a grande variedade de tipos de mensagem que podem ser transmitidos através deste protocolo. Em relação à eficiência, esta aplicação requer o envio

de bastantes segmentos de controlo (como estabelecimento de ligação, mudança de diretoria, etc). Tudo isso origina overhead desnecessário e que irá aumentar o rácio entre o número de bytes de dados que são transmitidos e o número de bytes transmitidos no total, reduzindo a eficiência.

TFTP Como vimos na pergunta 3, este protocolo corre em cima do protocolo UDP do nível de transporte. Como esse protocolo da camada 4 não tem mecanismos de controlo de erros, tem de ser a camada aplicacional a implementar estes serviços. Para tal, usa ACKS, que permitem ao emissor verificar a chegada de determinada mensagem ao destino. Este protocolo (TFTP) não fornece qualquer tipo de encriptação de dados. Como tal, a aplicação que o usa não permite transferências de ficheiros de forma segura. Para além de correr em cima do UDP, que por si só já é um protocolo bastante simples da camada de transporte, a transferência de dados através desta aplicação requer normalmente apenas a identificação de mais dois campos na camada aplicacional: o "opcode" (Read, Write, Data, Ack, Error) e um número que identifica o bloco (semelhante ao campo sequence number no tcp). Outros campos podem ser utilizados, dependendo do "opcode'. Por ser uma ligação mais simples, com menos segmentos transmitidos e por correr sobre UDP, esta ligação apresenta um menor overhead de transporte.

HTTP O protocolo HTTP, tal como o FTP e o SFTP, utiliza o protocolo TCP, protocolo da camada de transporte. Como tal, é fiável na transmissão dos dados. Contudo, apresenta algumas debilidades a nível da segurança pois não há qualquer tipo de encriptação dos dados e o conteúdo do ficheiro pode ser visualizado antes de chegar ao recetor. Para além disso, este protocolo não usa qualquer tipo de sistema de autenticação. Quanto à eficiência, podemos verificar que, para transmissões mais curtas, a percentagem de overhead não é tão elevada quanto o FTP, por exemplo, pois são transmitidos menos segmentos entre as duas partes. Contudo, à medida que aumentamos o tamanho do ficheiro, o número de segmentos trocados cresce bastante, aumentando o overhead da transmissão (mais informação de controlo é trocada). Para além disso, de acordo com a norma (rfc 2616), o http possui diversos tipos de campos adicionais que aumentam a complexidade (e o overhead) deste tipo de ligações. Por exemplo, é transmitida a data, o estado da conexão, tipo da mensagem (request ou response), entre muitos outros. Ainda assim, tendo por base os dados da tabela do exercício 5, podemos verificar que, para o caso da transferência de ficheiros pequenos (file1), o overhead do ftp é superior, uma vez que este assegura outro tipo de funcionalidade que o http não permite (por exemplo, autenticação).

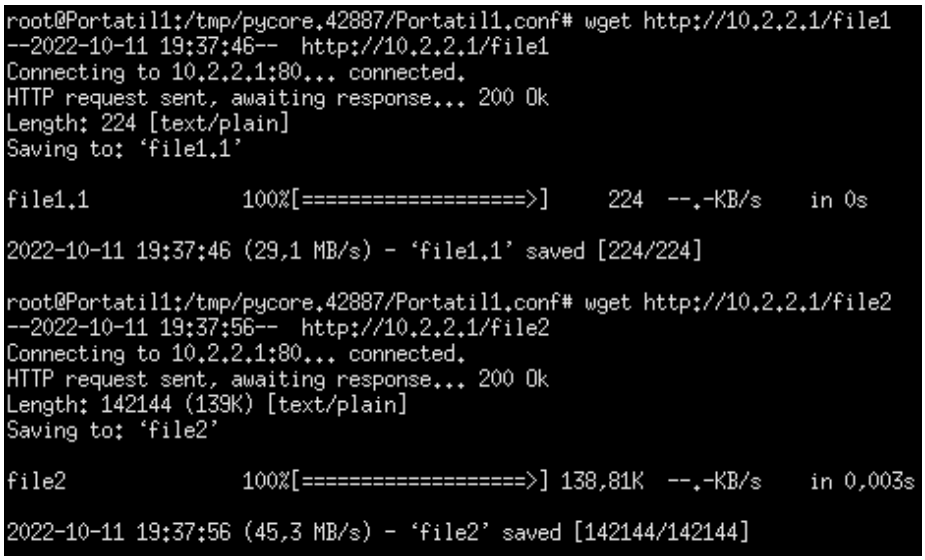

Figura 12: Transmissão HTTP dos ficheiros 1 e 2 para o Portatil1

1.5 Exercício 5

Com base no trabalho realizado, construa uma tabela informativa identificando, para cada aplicação executada (ping, traceroute, telnet, ftp, tftp, wget/lynx, nslookup, ssh, etc.), qual o protocolo de aplicação, o protocolo de transporte, a porta de atendimento e o overhead de transporte.

Aplicação	Protocolo de Aplicação	Protocolo de Transporte	Porta	*Overhead* de Transporte
ping	-	-	-	-
traceroute	-	-	-	-
telnet	TELNET	TCP	23	10,8%
ftp	FTP	TCP	21	68,0%
tftp	TFTP	UDP	69	8,2%
wget/lynx	HTTP	TCP	80	31,9%
nslookup	DNS	UDP	53	10,4%
ssh	SSH	TCP	22	22,1%

O valor do comprimento do cabeçalho dos pacotes que usam como protocolo de transporte TCP é, por norma, 20 *bytes*, uma vez que o campo "options" está quase sempre vazio. Contudo, este valor pode ir até 60 *bytes* consoante o número de opções. Acerca dos pacotes que usam como protocolo de transporte UDP, o tamanho do cabeçalho é fixo - 8 *bytes*.

Para o cálculo do valor de *overhead*, em percentagem, utilizamos a seguinte expressão: $\sum_{i=1}^{n} \frac{t_i}{t_i+a_i}$, em que t_i representa o tamanho do cabeçalho de transporte, a_i o tamanho dos dados e n o número de pacotes capturados.

Para as tranferências nas aplicações FTP, TFTP, WGET e SFTP usamos o ficheiro "file1" (224 *bytes*), menos extenso que o "file2" (138,81 *Kbytes*), o que leva a contrastes maiores na percentagem de *overhead*, uma vez que a maior parte dos dados são usados para controlo de ligação. Para a transferência por TELNET utilizamos o comando "telnet towel.blinkenlights.nl".

A eficiência das aplicações está relacionada com o *overhead* de transporte dos pacotes. Usualmente, quanto maior for o *overhead* menor será a eficiência da aplicação.

Através das capturas realizadas para o preenchimento da tabela, pudemos concluir que os valores das aplicações com protocolo de transporte UDP são menores que os das aplicações que utilizam TCP. Para além disso, os valores mais altos de *overhead* encontram-se em conexões que utilizam bastantes segmentos de controlo, como por exemplo nas transferências por FTP e HTTP, em conformidade com a questão 4.