

UNIVERSIDADE DO MINHO
MESTRADO EM ENGENHARIA INFORMÁTICA

Engenharia de Serviços em Rede
TP1: Streaming de áudio e vídeo a pedido e em tempo real
Grupo 61

Rui Guilherme Monteiro (PG50739) Rui Moreira (PG50736)
Rodrigo Rodrigues (PG50726)

Ano Letivo 2022/2023



Conteúdo

0.1	Introdução	3
0.2	Questões e Respostas	4
0.2.1	Etapa 1. Streaming HTTP simples sem adaptação dinâmica de débito	4
0.2.2	Etapa 2. Streaming adaptativo sobre HTTP (MPEG-DASH)	15
0.2.3	Etapa 3. <i>Streaming RTP/RTCP unicast</i> sobre UDP e <i>multicast</i> com anúncios <i>SAP</i>	19
1	Conclusão	23

0.1 Introdução

Este relatório é relativo ao primeiro trabalho prático da Unidade Curricular de Engenharia de Serviços em Rede cujo objetivo é experimentar várias soluções de streaming a pedido e em tempo real usando o emulador CORE como bancada experimental. Adicionalmente, pretende-se explorar as opções disponíveis em termos de pilha protocolar e diversos formatos multimédia, bem como o seu empacotamento.

Para concretizar os objetivos deste trabalho prático utilizar-se-á a ferramenta *FFmpeg* para criar um pequeno vídeo essencial para as 3 etapas do trabalho.

Ao longo das diferentes etapas do trabalho interessa-nos estudar o impacto que cada solução de streaming tem em termos de tráfego na rede. Na primeira etapa pretende-se fazer *streaming* por HTTP, onde o VLC atua como servidor e cliente (outros clientes são firefox e ffplay). Na segunda etapa faz-se *streaming* com adaptação de débito pelo que o servidor é substituído pelo ffmpeg. Na última etapa, utilizar-se-ão protocolos de streaming sobre UDP.

0.2 Questões e Respostas

0.2.1 Etapa 1. Streaming HTTP simples sem adaptação dinâmica de débito

Questão 1

Capture três pequenas amostras de trágefo no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (VLC e Firefox) e com 3 clientes (VLC, Firefox e ffplay). Identifique a taxa em bps necessária (usando o ffmpeg -i videoA.mp4 e/ou o próprio wireshark), o encapsulamento usado e o número total de fluxos gerados. Comente a escalabilidade da solução. Ilustre com evidências da realização prática do exercício (ex: capturas de ecrã).

1.1) Para a realização deste exercício, começamos por construir uma topologia base no CORE, com um servidor, 3 portáteis, 2 switches e dois routers, tal como representado na figura abaixo.

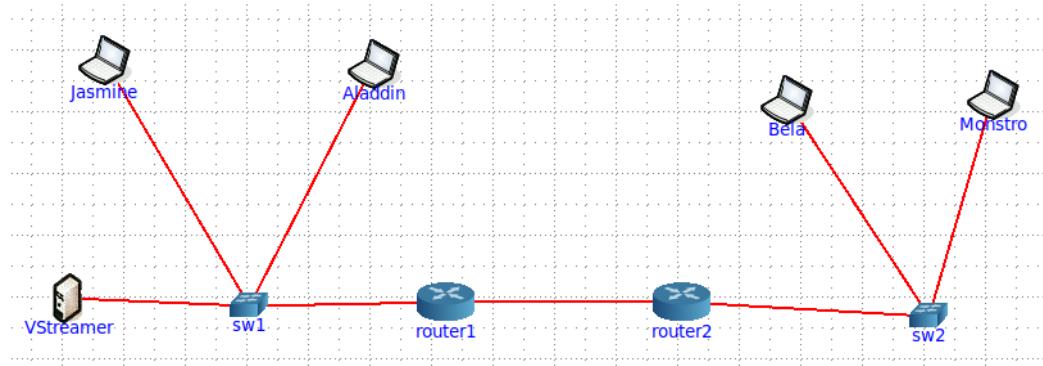


Figura 1: Topologia base no CORE.

1.2) De seguida testou-se a conectividade da rede, com recurso ao comando *ping*.

Após testes de conectividade, via comando *ping*, verificou-se que os dispositivos da rede são alcançáveis.

```
root@Aladdin:/tmp/pycore_36039/Aladdin.conf# ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=63 time=0.057 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=63 time=0.036 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=63 time=0.034 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=63 time=0.034 ms
64 bytes from 10.0.2.1: icmp_seq=5 ttl=63 time=0.034 ms
64 bytes from 10.0.2.1: icmp_seq=6 ttl=63 time=0.036 ms
```

Figura 2: *Ping* de Alladin para Router1.

```

root@router1:/tmp/pycore_36039/router1.conf# ping 10.0.2.21
PING 10.0.2.21 (10.0.2.21) 56(84) bytes of data.
64 bytes from 10.0.2.21: icmp_seq=1 ttl=63 time=0.045 ms
64 bytes from 10.0.2.21: icmp_seq=2 ttl=63 time=0.037 ms
64 bytes from 10.0.2.21: icmp_seq=3 ttl=63 time=0.034 ms
64 bytes from 10.0.2.21: icmp_seq=4 ttl=63 time=0.034 ms
64 bytes from 10.0.2.21: icmp_seq=5 ttl=63 time=0.034 ms
64 bytes from 10.0.2.21: icmp_seq=6 ttl=63 time=0.034 ms
64 bytes from 10.0.2.21: icmp_seq=7 ttl=63 time=0.035 ms
64 bytes from 10.0.2.21: icmp_seq=8 ttl=63 time=0.038 ms

```

Figura 3: Ping de Router1 para Monstro.

1.3) No servidor VStreamer, fazer streaming por HTTP do ficheiro videoA.mp4.

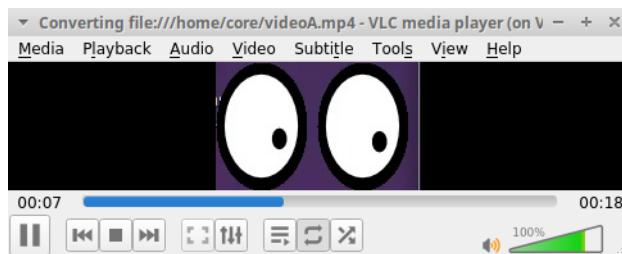


Figura 4: Streaming por HTTP no servidor VStreamer.

1.4) No portátil Jasmine, configurar o DISPLAY e colocar um segundo VLC a funcionar agora como cliente.



Figura 5: VLC no portátil Jasmine como cliente.

1.5) Capturamos agora uma amostra de tráfego no link de saída do servidor, via Wireshark, com 1 cliente (VLC).

1.6) De seguida, construimos uma página HTML, video.html, com um conteúdo simples usando a TAG <vídeo>.

```

<!DOCTYPE html>
<html>
  <head>
    <title> Streaming ESR </title>

```

```

</head>
<body>
    <h1> Streaming ESR: Etapa 1 </h1>
    <video controls autoplay>
        <source src="http://10.0.0.10:8080">
    </video>
</body>
</html>

```

1.7) No portátil Bela, abrimos uma *bash*, configuramos o *DISPLAY* e executamos o *firefox*. Abrimos a página *video.html* criada anteriormente e verificamos a reprodução correta do vídeo.

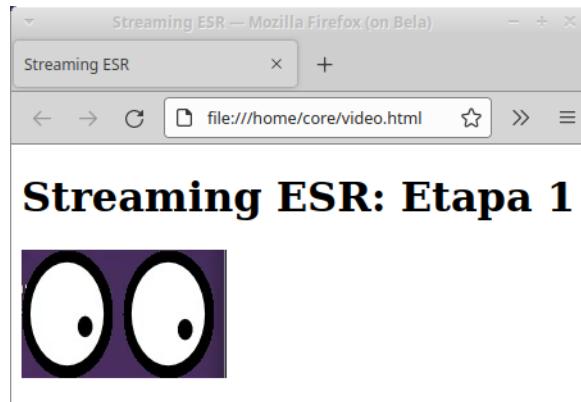


Figura 6: Bela: Vídeo em reprodução na página *video.html* criada.

1.8) Procedemos à recolha de mais uma amostra de tráfego, neste caso com 2 clientes (VLC e Firefox).

1.9) De modo a testar uma solução de *streaming* com 3 clientes (VLC, Firefox e ffplay), abrimos uma *bash* no portátil Monstro, configuramos o *DISPLAY* e usamos o comando ffplay *http://<endereço-IP>:8080/* como terceiro cliente da *stream* de vídeo.

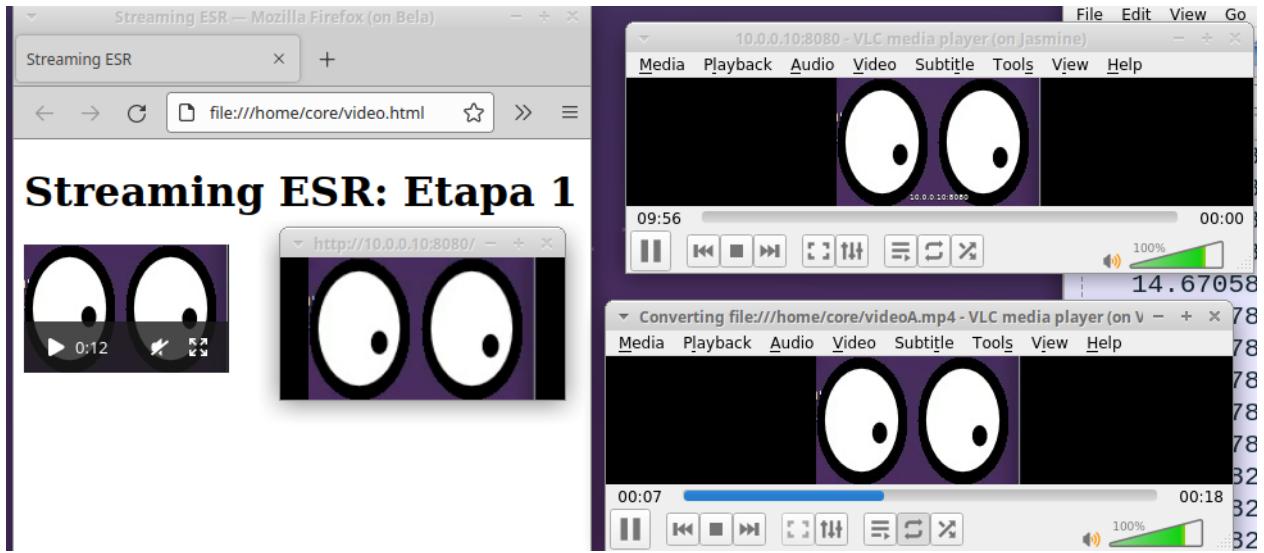


Figura 7: *Streaming* com 3 clientes

1.9) Identificação da taxa em bps necessária através do comando referido no enunciado. Como é possível verificar pela figura abaixo, o valor teórico da taxa de bits necessária é de

$$8kb/s \approx 64000bps$$

```
core@xubuncore:~$ ffmpeg -i videoA.mp4
ffmpeg version 4.2.7-0ubuntu0.1 Copyright (c) 2000-2022 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.4.0-1ubuntu1~20.04.1)
configuration: --prefix=/usr --extra-version=0ubuntu0.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresample --disable-filter=resample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libcodec2 --enable-libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librsvg --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-libzvbi --enable-lv2 --enable-omx --enable-openal --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-nvenc --enable-chromaprint --enable-frei0r --enable-libx264 --enable-shared
  libavutil      56. 31.100 / 56. 31.100
  libavcodec     58. 54.100 / 58. 54.100
  libavformat    58. 29.100 / 58. 29.100
  libavdevice    58.   8.100 / 58.   8.100
  libavfilter     7. 57.100 /  7. 57.100
  libavresample   4.   0.  0 /  4.   0.  0
  libswscale       5.  5.100 /  5.  5.100
  libswresample   3.  5.100 /  3.  5.100
  libpostproc    55.  5.100 / 55.  5.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoA.mp4':
  Metadata:
    major_brand : isom
    minor_version : 512
    compatible_brands: isomiso2avc1mp41
    encoder : Lavf58.45.100
Duration: 00:00:18.05, start: 0.000000, bitrate: 10 kb/s
  Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 160x100, 8 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
    Metadata:
      handler_name : VideoHandler
At least one output file must be specified
```

Figura 8: `ffmpeg -i videoA.mp4`

Fazendo uso da ferramenta *Protocol Hierarchy* do módulo *Statistics* do *Wireshark*, e aplicando o filtro *tcp.stream == 0/1/2* é possível determinar quantos Bits/s correspondem ao HTTP em cada um dos casos (1/2/3 clientes). Podemos observar que a taxa real necessária para a transmissão do vídeo foi na verdade

$$\text{Bits/s}(1\text{cliente}) = 12\text{kb/s}$$

Para a captura com 1 cliente (VLC). Este valor difere do valor teórico calculado (8kb/s) (Figura 0.2.1) e pode ser justificado pela ocorrência de perdas de pacotes na transmissão.

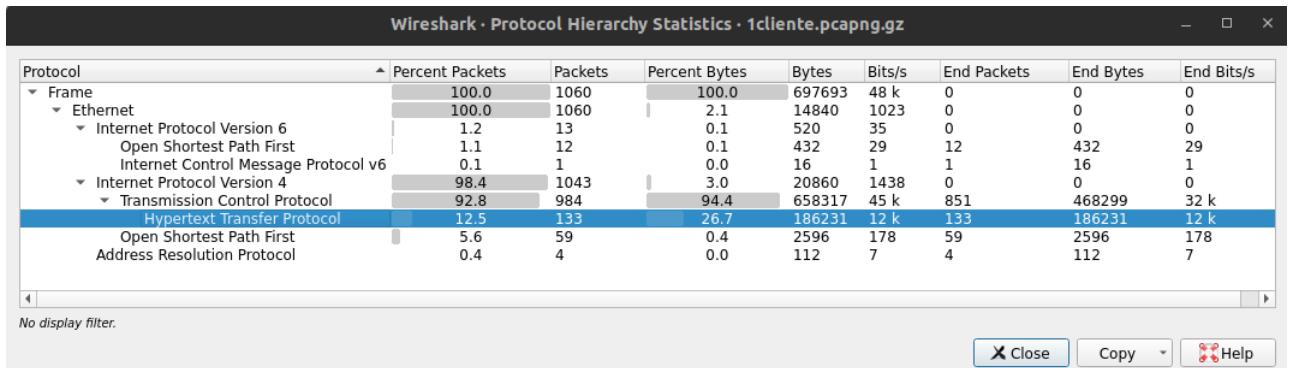


Figura 9: Taxa bps real na transmissão com 1 cliente

$$\text{Bits/s}(2\text{clientes}) = 12k + 12k = 24k$$

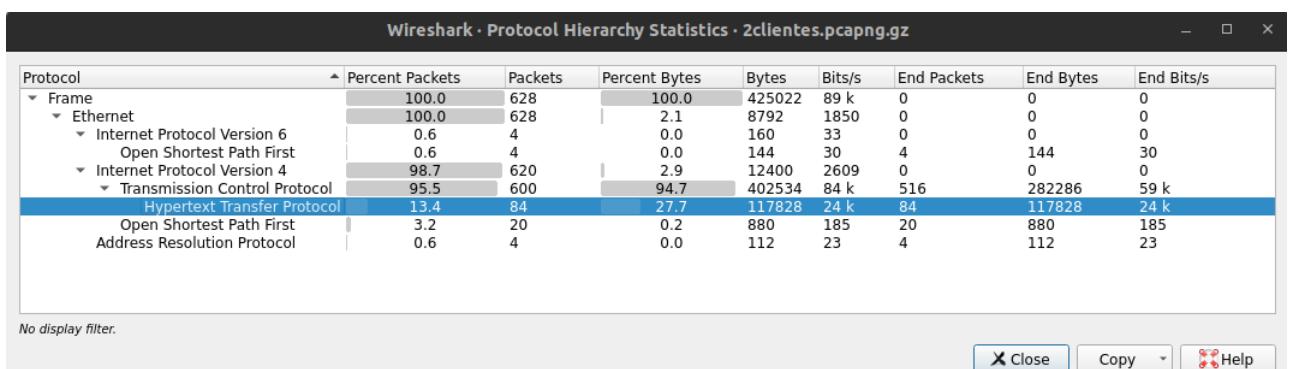


Figura 10: Taxa bps real na transmissão com 2 clientes

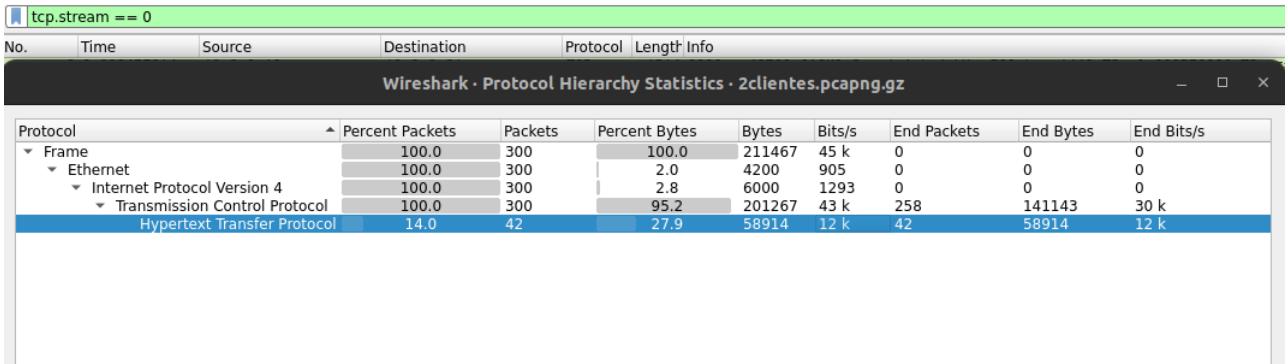


Figura 11: Taxa bps real na transmissão com 2 clientes, usando o filtro `tcp.stream == 0`

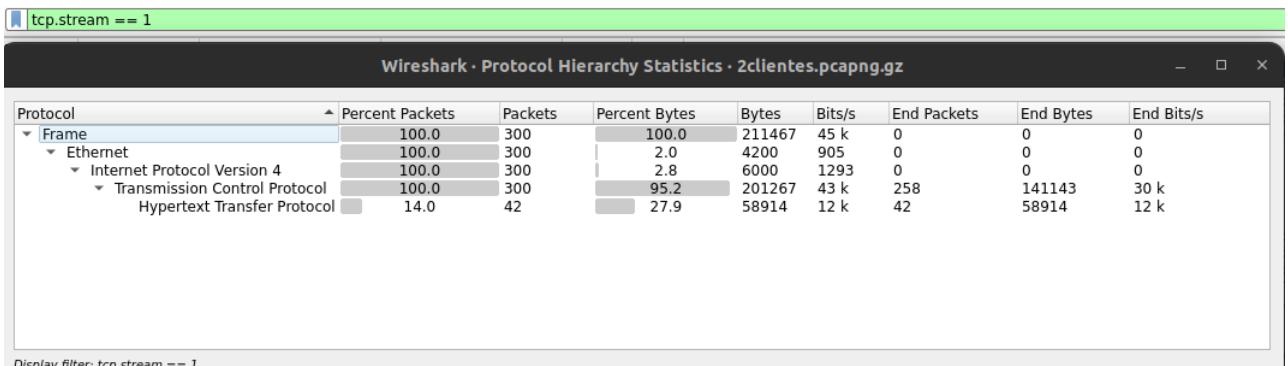


Figura 12: Taxa bps real na transmissão com 2 clientes, usando o filtro `tcp.stream == 1`

$$Bits/s(3\text{clientes}) = 13k + 13k + 13k = 39k$$

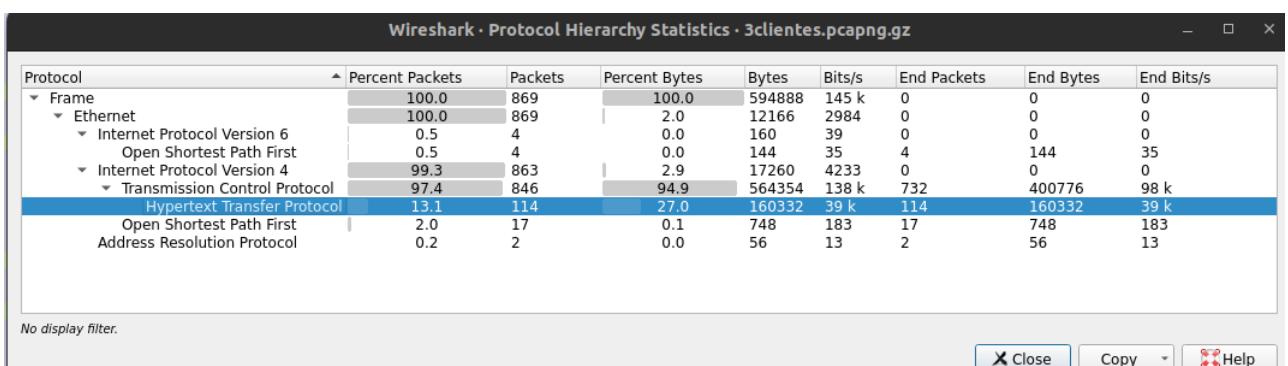


Figura 13: Taxa bps real na transmissão com 3 clientes

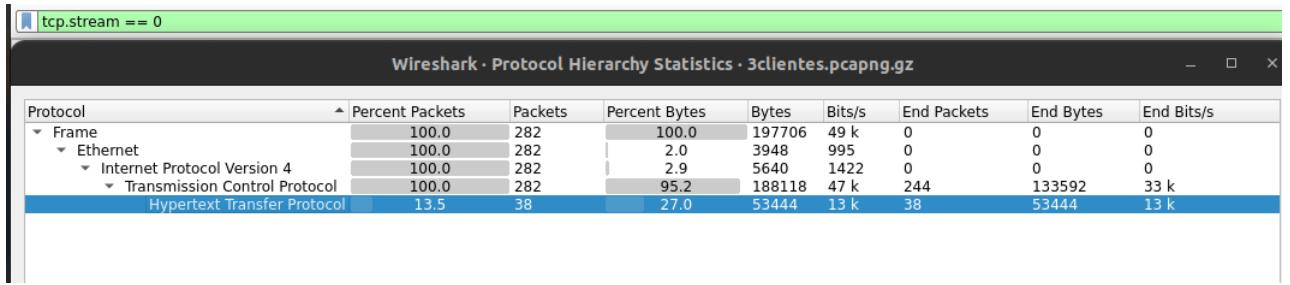


Figura 14: Taxa bps real na transmissão com 3 clientes, usando o filtro `tcp.stream == 0`

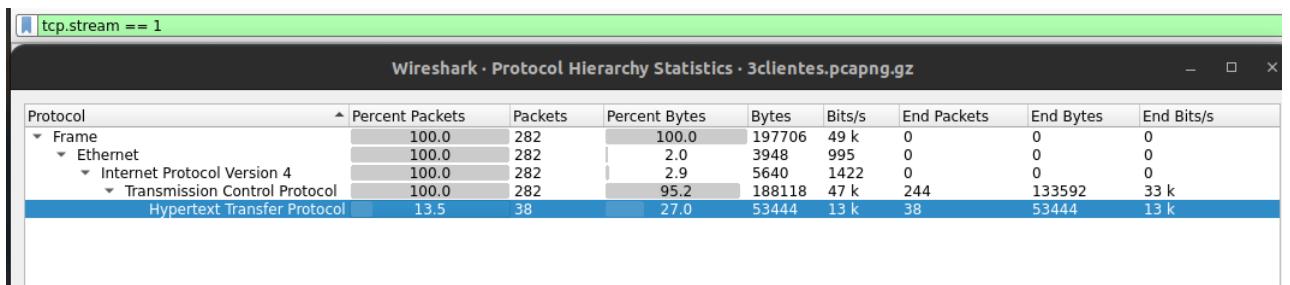


Figura 15: Taxa bps real na transmissão com 3 clientes, usando o filtro `tcp.stream == 1`

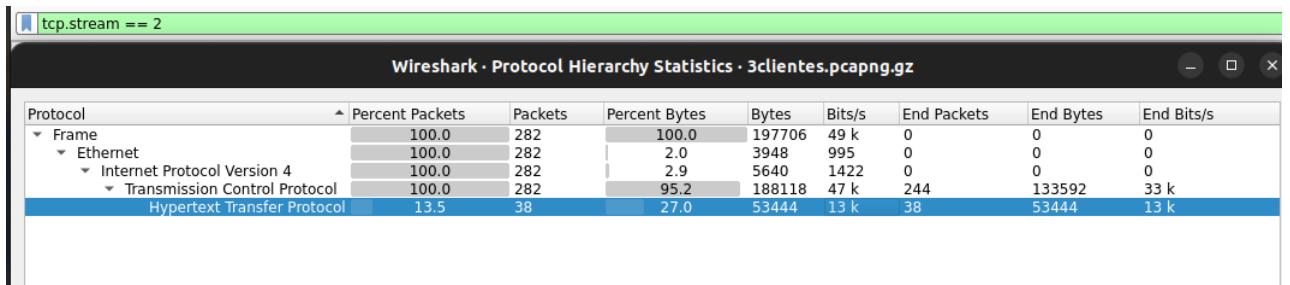


Figura 16: Taxa bps real na transmissão com 3 clientes, usando o filtro `tcp.stream == 2`

Partindo da análise de tramas capturadas pelo *wireshark*, nos 3 casos pelas Figuras 17 18 e 19 respetivamente, conclui-se que o encapsulamento dos pacotes transmitidos é um processo que ocorre em diferentes níveis da pilha protocolar: camada de transporte (TCP), camada de rede (IPv4), camada de aplicação (HTTP), e camada de ligação de dados (protocolo *Ethernet*).

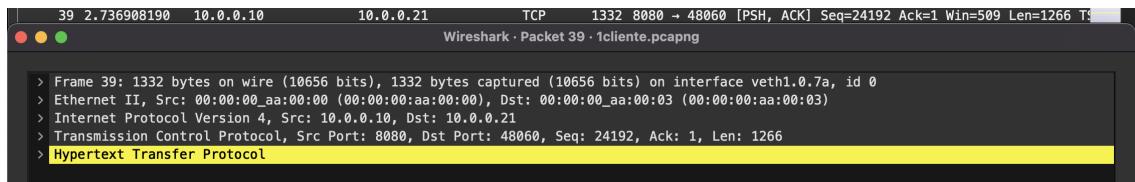


Figura 17: Encapsulamento - 1 cliente

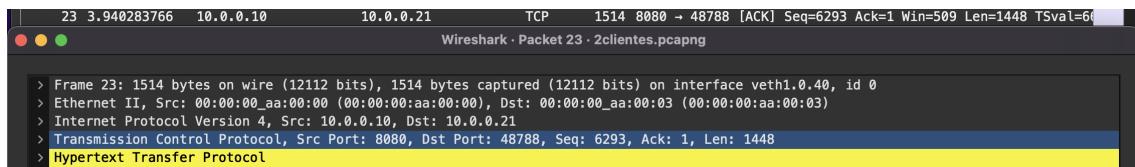


Figura 18: Encapsulamento - 2 clientes

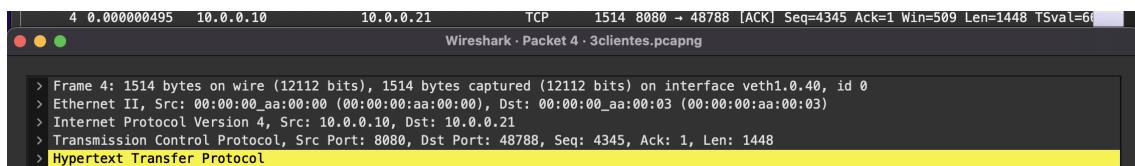


Figura 19: Encapsulamento - 3 clientes

O cálculo do número total de fluxos gerados a partir de uma captura *Wireshark* tem em conta os seguintes parâmetros dos pacotes:

- IP Origem, IP Destino
- Protocolo
- Porta Origem, Porta Destino

Consultando Statistics -> Conversations pode-se visualizar as conversas de rede (tráfego entre dois *endpoints* específicos).

Na solução de 1 cliente (VLC), foi gerado 1 fluxo, como é possível averiguar através do filtro *tcp.stream eq*

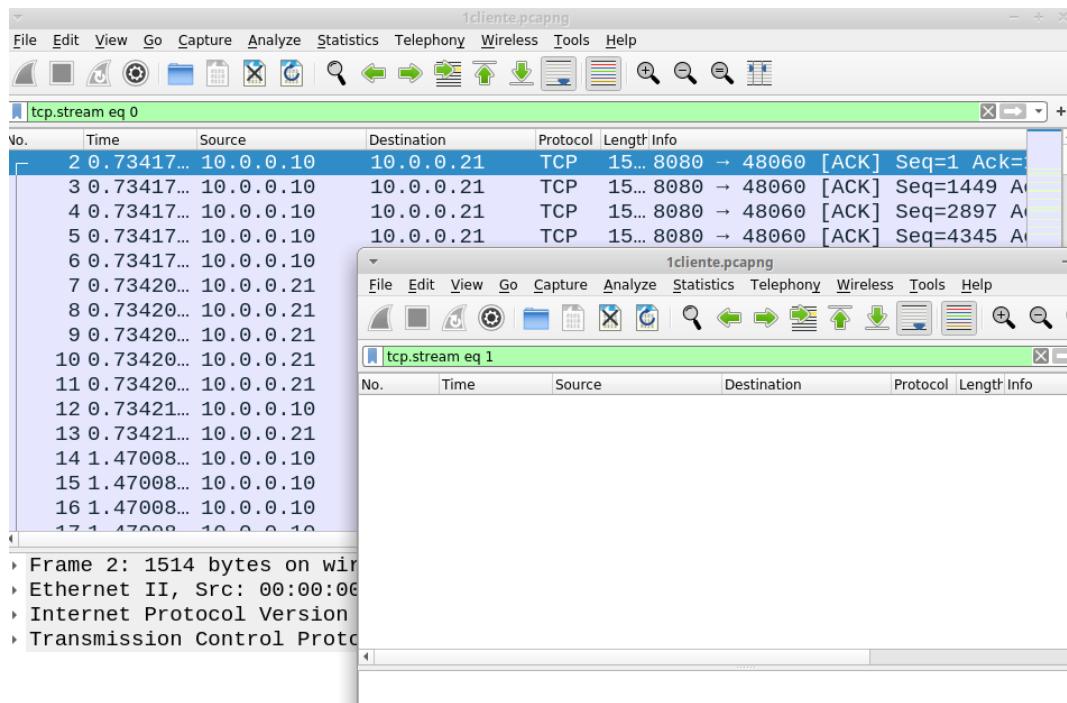


Figura 20: Fluxos gerados a partir da captura de tráfego com 1 cliente

Na solução de 2 clientes (VLC, Firefox), foram gerados 2 fluxos, correspondentes a cada um dos serviços VLC e Firefox.

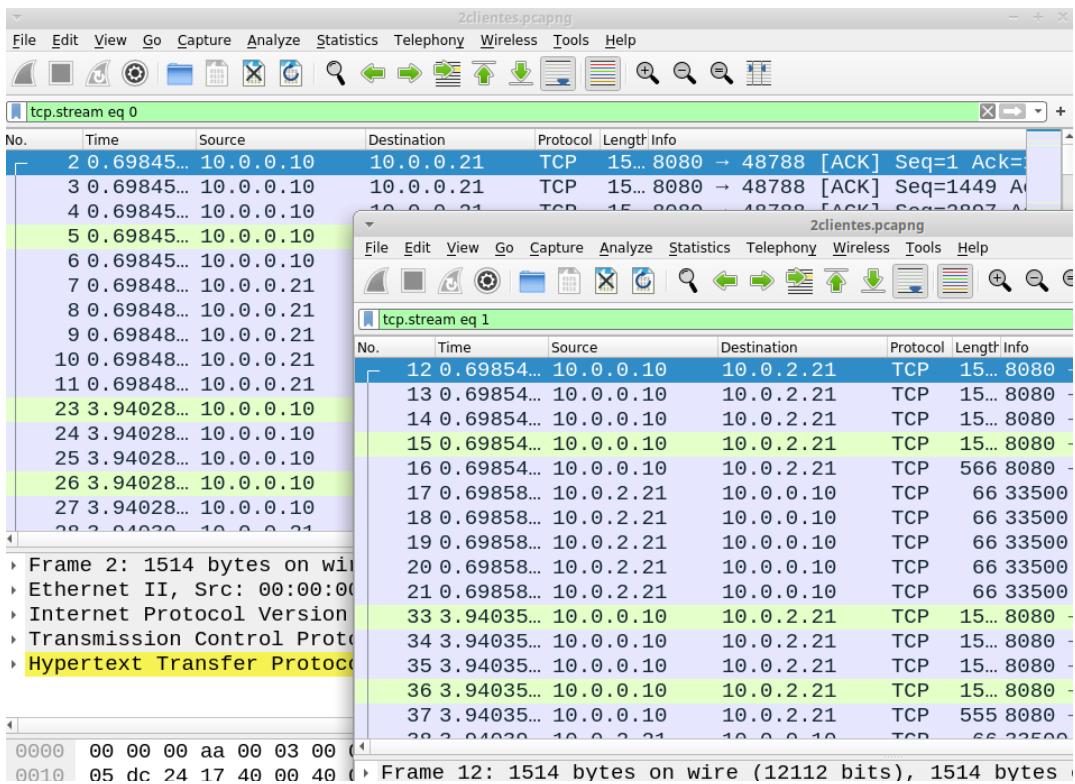


Figura 21: Fluxos gerados a partir da captura de tráfego com 2 clientes

Na solução de 3 clientes (VLC, Firefox, ffplay), foram gerados 3 fluxos correspondentes a cada um dos serviços VLC, Firefox e ffplay.

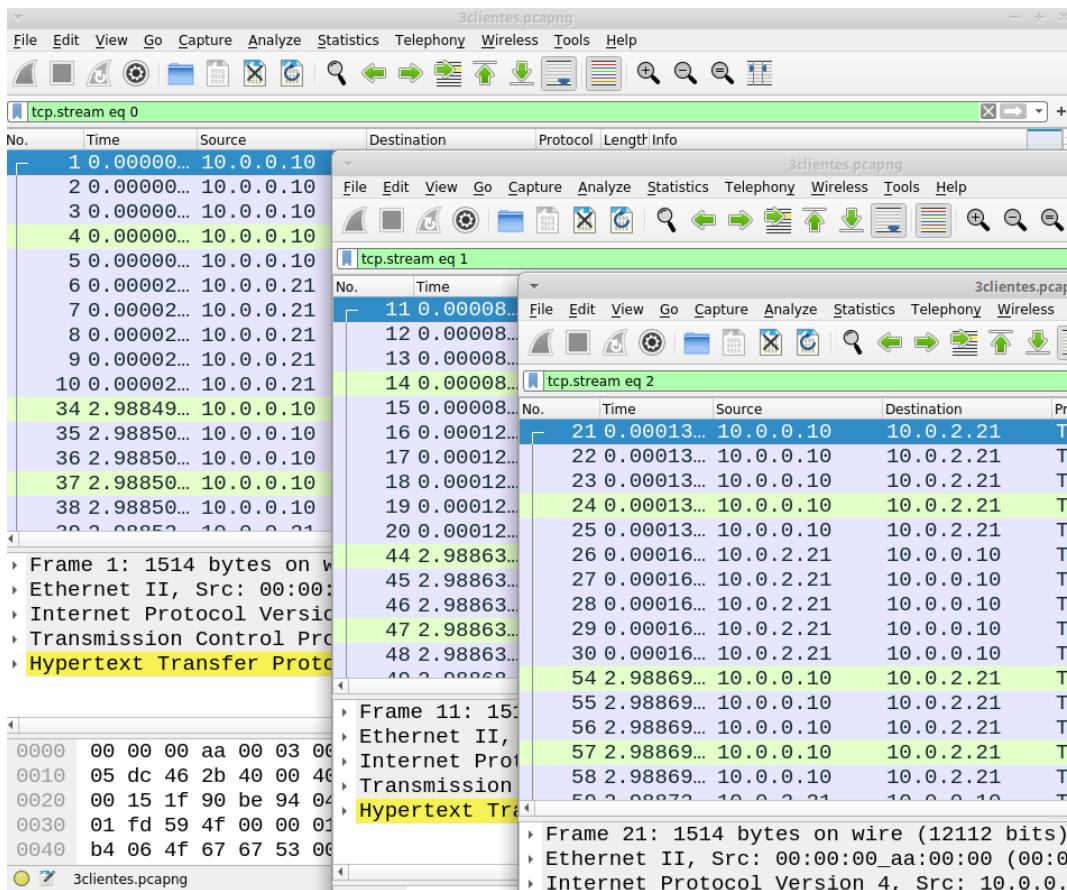


Figura 22: Fluxos gerados a partir da captura de tráfego com 3 clientes

	VLC	Firefox	ffplay	Total
1 cliente	12k			12k
2 clientes	12k	12k		24k
3 clientes	13k	13k	13k	39k

Tabela 1: Taxa em Bps necessária para os 3 casos

Por observação das capturas efetuadas com diferentes clientes, conclui-se que a solução de adicionar *streams* é pouco escalável na medida em que o servidor envia resposta a pedidos individualmente a cada cliente, pelo que o aumento do número de clientes implicaria um crescimento considerável do envio de tramas para clientes, mesmo que hajam pedidos semelhantes.

0.2.2 Etapa 2. Streaming adaptativo sobre HTTP (MPEG-DASH)

Questão 2

Diga qual a largura de banda necessária, em bits por segundo, para que o cliente de *streaming* consiga receber o vídeo no *firefox* e qual a pilha protocolar usada neste cenário.

A informação sobre a largura de banda necessária, em bits por segundo, para que o cliente de *streaming* consiga receber o vídeo no *firefox* está presente no ficheiro *video_manifest.mpd*. Esta largura de banda necessária depende da resolução do vídeo.

```
<Representation id="1" mimeType="video/mp4" codecs="avc3.64080c" width="160" height="100" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="105054">
<Representation id="2" mimeType="video/mp4" codecs="avc3.640014" width="320" height="200" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="238842">
<Representation id="3" mimeType="video/mp4" codecs="avc3.64001e" width="640" height="400" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="605145">
```

Figura 23: Algumas linhas do ficheiro *video_manifest.mpd*

Como podemos observar na Figura 23, para que o cliente consiga receber o vídeo de menor resolução (160x100), é necessário uma largura de banda de 105054 bps. Para o vídeo de resolução intermédia (320x200), é preciso uma largura de banda de 238842 bps. Por fim, para o vídeo com maior resolução (640x400) é necessário uma largura de banda de 605145 bps.

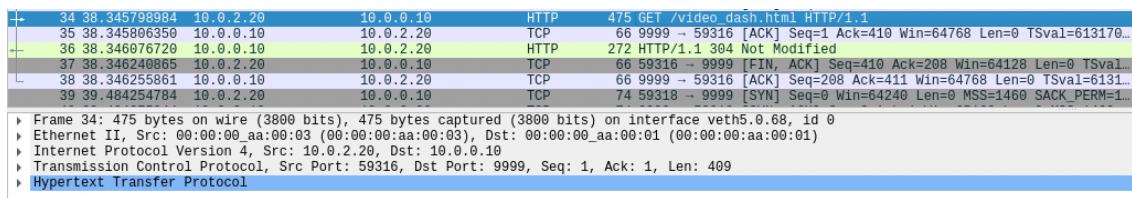


Figura 24: Pilha Protocolar

Em relação à pilha protocolar usada neste cenário, como podemos ver na Figura 24, na camada de ligação de dados é usado o protocolo *Ethernet*, na camada de rede o protocolo *IP*, na camada de transporte o protocolo *TCP* e na camada de aplicação *HTTP*.

Questão 3

Ajuste o débito dos links da topologia de modo que o cliente no portátil Bela exiba o vídeo de menor resolução e o cliente no portátil *Alladin* exiba o vídeo com mais resolução. Mostre evidências.

Para que o portátil Bela exiba o vídeo de menor resolução é necessário restringir o link à largura de banda. Como vimos na questão anterior, a largura de banda necessária para transmitir o vídeo com menor resolução é 105054 bps, basta que o link que liga o *switch* ao portátil *Bela* seja limitado a um débito de 150000bps (150.00kbps).

Para que o portátil *Alladin* exiba o vídeo com maior resolução, não é necessário limitar a capacidade do link.

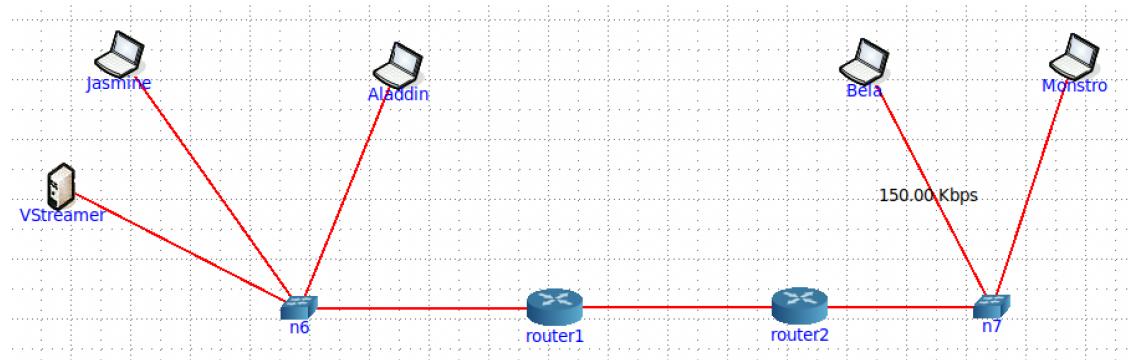


Figura 25: Topologia com as restrições

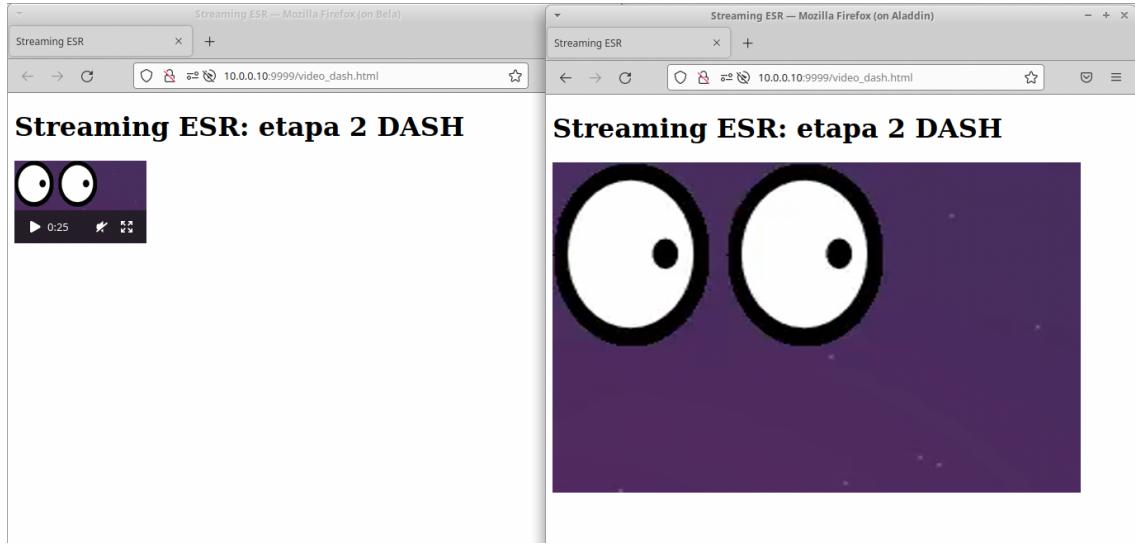


Figura 26: Comparação da Resolução das *streams*

Como podemos ver na Figura 26, obtivemos o pretendido, portátil *Bela* a exibir o vídeo de menor resolução (160x100) e o portátil *Alladin* a exibir o vídeo de maior resolução (640x400).

Questão 4

Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado.

O *DASH* (*Dynamic Adaptive Streaming over HTTP*) é uma técnica de *streaming* por HTTP que permite a transmissão dinâmica e adaptativa às condições da rede como por exemplo a largura de banda do *link*. O modo de procedimento do *DASH* passa por verificar a largura de banda do *link* de transmissão e avaliar qual a melhor resolução do vídeo a ser transmitido nesse *link*, a partir do ficheiro "video_manifest.mpd", que possui informação sobre cada resolução disponível do vídeo, como largura de banda necessária do link para transmitir o vídeo nessa resolução específica. Desta forma, durante o *stream* do vídeo, o *DASH* vai avaliando a qualidade de transmissão e conectividade e alterando a resolução do vídeo conforme essas circunstâncias.

Neste caso concreto, com o auxílio de uma captura de *Wireshark*,

conseguimos visualizar o funcionamento do *DASH*.

262	19.	842075457	10.0.0.21	10.0.0.10	HTTP	399 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
1704	19.	930749722	10.0.0.21	10.0.0.10	HTTP	401 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
3182	20.	012510934	10.0.0.21	10.0.0.10	HTTP	401 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
5210	20.	093270188	10.0.0.21	10.0.0.10	HTTP	402 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
6852	20.	173261318	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
8389	20.	276137296	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
10083	20.	353190238	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
11776	20.	437392488	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
13329	20.	503350869	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
14931	20.	577710370	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
16381	20.	660756025	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
18900	20.	741194452	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
19863	20.	801637978	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
21558	20.	870279692	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
23041	20.	968963962	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
24635	21.	058641097	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
26127	21.	152331915	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
27653	21.	229345162	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
29241	21.	332050057	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
30658	21.	403827191	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
32148	21.	502072909	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
33869	21.	602735288	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
35366	21.	700881921	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
36849	21.	768552877	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
38535	21.	832189065	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
40007	21.	896491227	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
41427	21.	979826108	10.0.0.21	10.0.0.10	HTTP	403 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
43214	22.	054643859	10.0.0.21	10.0.0.10	HTTP	404 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
44987	22.	136783206	10.0.0.21	10.0.0.10	HTTP	405 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
46606	22.	226368401	10.0.0.21	10.0.0.10	HTTP	405 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
48215	22.	324669124	10.0.0.21	10.0.0.10	HTTP	405 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
49809	22.	400030752	10.0.0.21	10.0.0.10	HTTP	405 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
54416	22.	404527054	10.0.0.21	10.0.0.10	HTTP	405 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1

Figura 27: Portátil Alladin

Como podemos ver na Figura 27, no portátil *Alladin*, onde não existia limitação do débito do link, podemos ver que o *stream* foi inicializado com o vídeo de maior resolução (640x400). Ao longo da *stream* manteve-se constante e manteve-se a fazer o *streaming* do vídeo com maior resolução.

16	12.	044264984	10.0.2.20	10.0.0.10	HTTP	379 GET /favicon.ico HTTP/1.1
25	12.	598322020	10.0.2.20	10.0.0.10	HTTP	399 GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
145	16.	736887187	10.0.2.20	10.0.0.10	HTTP	400 GET /videoB_320_200_500k_dash.mp4 HTTP/1.1
186	17.	485588606	10.0.2.20	10.0.0.10	HTTP	399 GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
82569	50.	887765118	10.0.2.20	10.0.0.10	HTTP	400 GET /videoB_160_100_200k_dash.mp4 HTTP/1.1

Figura 28: Portátil Bela

Já no portátil Bela, onde existe limitação do débito do link para que exibisse o vídeo de menor resolução, podemos visualizar na Figura 28, que começou pelo vídeo de maior resolução (640x400), mas viu que não tinha débito suficiente, tentou mudar para o vídeo de resolução intermédia (320x200), continuou a achar que não era suficiente e mudou para o vídeo de menor resolução (160x100), sendo esta a resolução que acaba por ser transmitida até ao fim da *stream*.

Isto aconteceu pois o *DASH* verificou que a conectividade e/ou a transmissão na rede não eram suficientes para manter a transmissão, na íntegra, da resolução mais alta (640*400) e por isso baixa resolução do vídeo para que o utilizador conseguisse ver o vídeo completo sem que note perdas de *frame* devido à falta de conexão.

0.2.3 Etapa 3. *Streaming RTP/RTCP unicast* sobre UDP e *multicast* com anúncios SAP

Questão 5

Compare o cenário aplicado com o cenário *multicast*. Mostre vantagens e desvantagens na solução ao nível da rede, no que diz respeito a escalabilidade (aumento do nº de clientes) e tráfego na rede. Tire as suas conclusões.

O teste de cenários de rede distintos, para *streaming* em *unicast* (apenas um *sender* e um *receiver*) e para *multicast* (envio de pacotes para um grupo de *hosts* na rede), permitiu efetuar uma análise de vantagens e desvantagens desta última ao nível da rede.

Em termos de **escalabilidade**, i.e aumento do número de clientes na rede, conclui-se que o serviço em *multicast* é mais escalável do que *unicast* pela sua capacidade de dar resposta mais eficiente na presença de um conjunto de clientes na rede, aproveitando melhor a largura de banda e evitando o envio de pacotes redundantes (vantagem a nível de **tráfego de rede**).

Contudo, o *streaming* em *multicast* revela-se altamente complexo em comparação com *unicast*, o que representa uma desvantagem por implicar maiores custos de overhead.

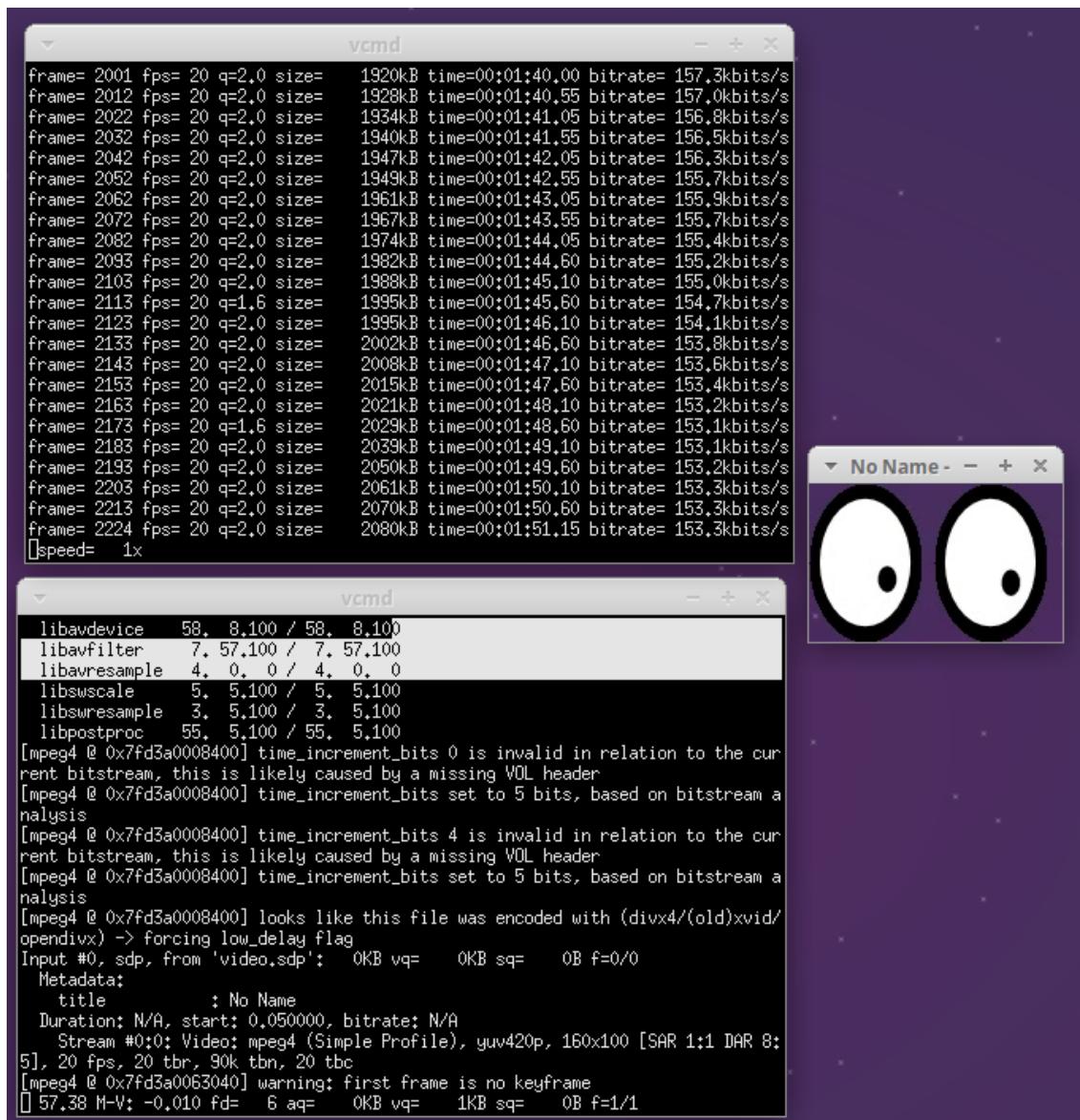


Figura 29: *Unicast com 1 cliente*

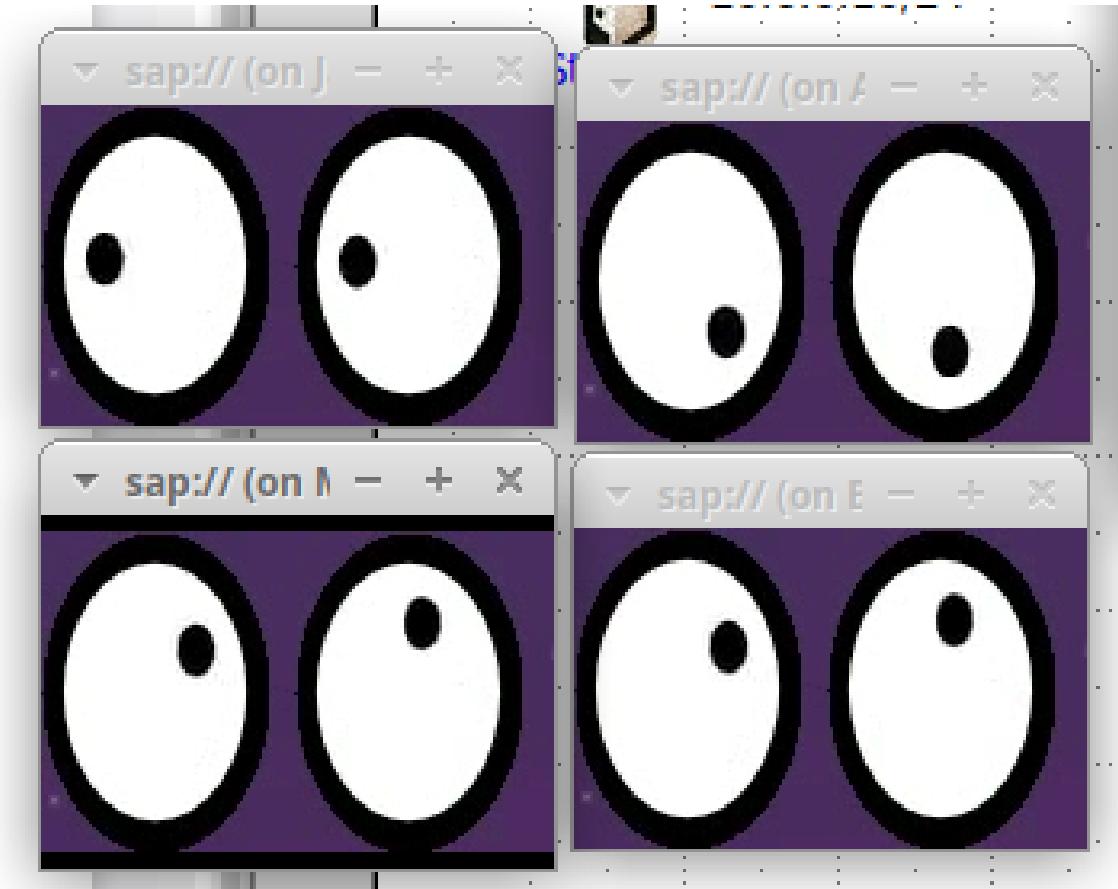


Figura 30: Multicast com 4 clientes

Através das capturas de tráfego *Wireshark* das transmissões unicast e multicast, observamos que a quantidade de dados transmitida em unicast (140k), relativos a 1 cliente, é superior à quantidade de dados transmitida em multicast (130k) relativa a 4 clientes. Comprova-se, tal como esperado, que em multicast existe melhor aproveitamento da largura de banda existente e portanto este serviço é mais escalável.

Wireshark - Protocol Hierarchy Statistics - unicast.pcapng								
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	341	100.0	229109	149 k	0	0	0
Ethernet	100.0	341	2.1	4774	3119	0	0	0
Internet Protocol Version 6	0.6	2	0.0	80	52	0	0	0
Open Shortest Path First	0.3	1	0.0	36	23	1	36	23
Internet Control Message Protocol v6	0.3	1	0.0	16	10	1	16	10
Internet Protocol Version 4	98.8	337	2.9	6740	4404	0	0	0
User Datagram Protocol	96.8	330	1.2	2640	1725	0	0	0
Data	96.5	329	93.6	214407	140 k	329	214407	140 k
ADwin configuration protocol	0.3	1	0.0	52	33	1	52	33
Open Shortest Path First	2.1	7	0.1	308	201	7	308	201
Address Resolution Protocol	0.6	2	0.0	56	36	2	56	36

Figura 31: Quantidade de dados transmitida com unicast

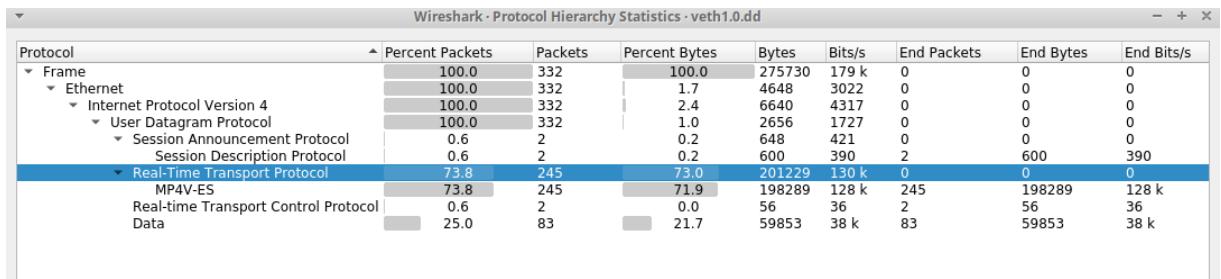


Figura 32: Quantidade de dados transmitida com multicast

Capítulo 1

Conclusão

Neste trabalho prático, foi possível consolidar o conhecimento adquirido ao longo das aulas teóricas desta unidade curricular. Na primeira etapa, foi feito *Streaming* HTTP simples. Apesar de ser uma opção muito popular, verificou-se que não é a opção mais viável pois não permite que a solução seja escalável. Já na segunda etapa, foi feito *Streaming* com débito adaptativo, tendo sido analisado o funcionamento do DASH e o seu papel no ajustamento da resolução dos vídeos de acordo com a qualidade da rede. Por último, na etapa 3, foi feito *Streaming* sobre UDP,tanto *unicast* como *multicast*, tendo-se comparando estas duas opções. Verificamos que a opção de *multicast* permite escalabilidade da rede, além de reduzir o seu tráfego.

Em suma, o grupo considera que cumpriu os objetivos deste trabalho prático, aprofundando os seus conhecimentos em *video streaming*.