

UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA

Engenharia de Serviços em Rede

Grupo 82

**TP1 : *Streaming* de áudio e vídeo a pedido e
em tempo real**

Danilo Queiroga Oliveira (PG46528)

Joana Maia Teixeira Alves (PG50457)

Vicente Gonçalves Moreira (PG50799)

Outubro 2022

Questões e Respostas

1 Streaming HTTP simples sem adaptação dinâmica de débito

Tarefas

Começamos por construir uma topologia com um servidor (*VStreamer*), 4 portáteis (*Jasmine*, *Alladin*, *Bela* e *Monstro*), 2 switches (*sw1* e *sw2*) e dois routers (*rt1* e *rt2*):

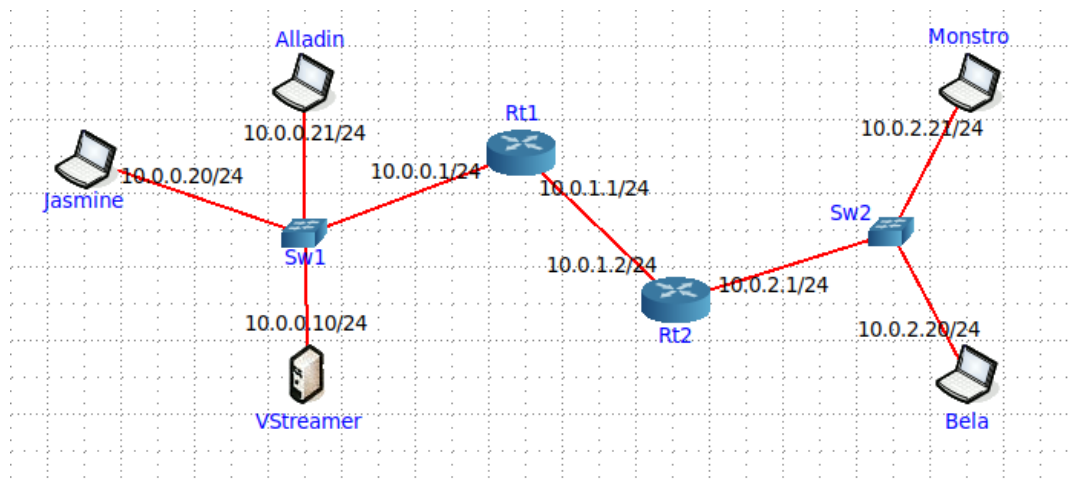


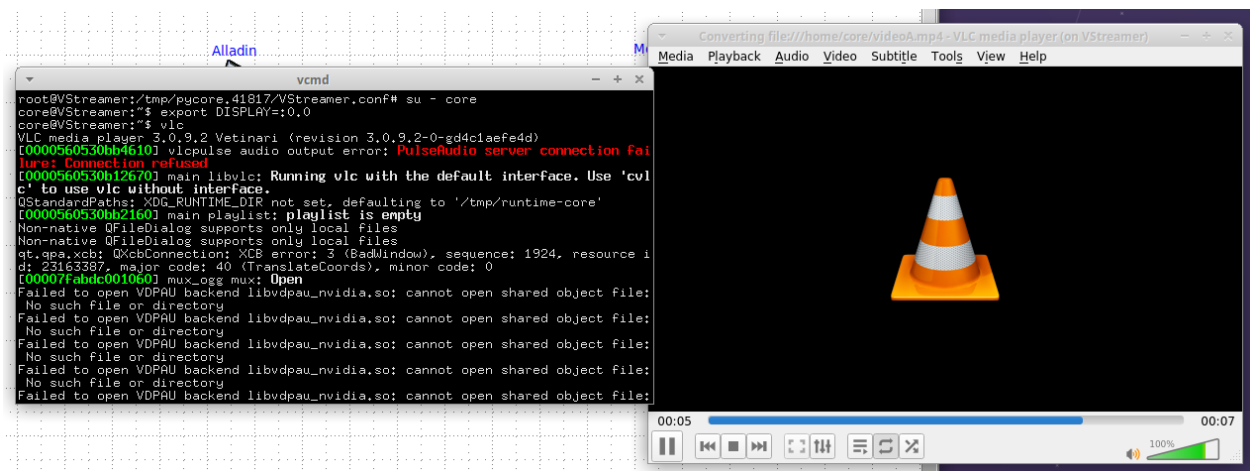
Figura 1: Topologia.

Para termos a certeza de que tudo funcionava bem, testamos a conectividade na rede através do comando *ping* a partir do portátil *Jasmine* até ao portátil *Bela*:

```
root@Jasmine:/tmp/pycore.41817/Jasmine.conf# ping 10.0.2.20
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=62 time=0.101 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=62 time=0.049 ms
64 bytes from 10.0.2.20: icmp_seq=3 ttl=62 time=0.047 ms
64 bytes from 10.0.2.20: icmp_seq=4 ttl=62 time=0.046 ms
64 bytes from 10.0.2.20: icmp_seq=5 ttl=62 time=0.046 ms
^C
--- 10.0.2.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 411ms
rtt min/avg/max/mdev = 0.046/0.057/0.101/0.021 ms
root@Jasmine:/tmp/pycore.41817/Jasmine.conf#
```

Figura 2: Comando *ping* de Jasmine para Bela.

Seguindo as indicações do ANEXO 2, a partir do servidor *VStreamer*, colocamos o VLC a fazer *streaming* por HTTP do ficheiro *videoA.mp4* com *transcoding* para "Video - Theora + Vorbis (Ogg)", assegurando que o *streaming* é servido em ciclo, sem parar:



De seguida, configuramos no portátil *Jasmine* um segundo VLC, mas agora a funcionar como cliente, assegurando que utilizamos o endereço IP do servidor *VStreamer* e a porta correspondente, obtendo o seguinte resultado. Também efetuamos simultaneamente uma captura de tráfego no servidor, para ser posteriormente analisada.

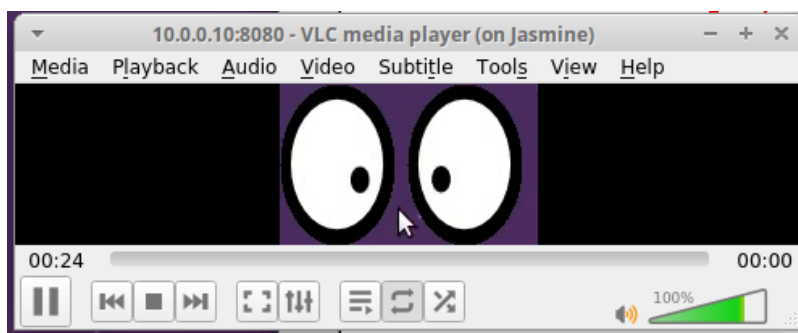
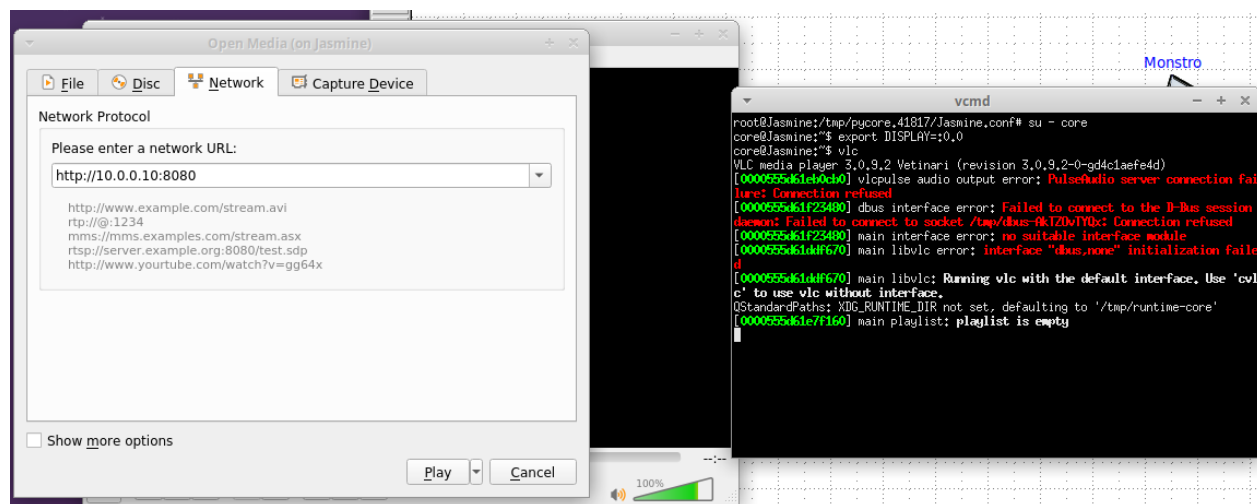
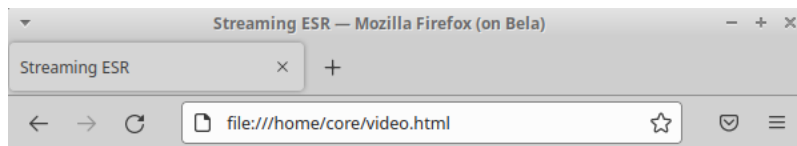


Figura 3: Configuração do portátil Jasmine e visualização da *stream*

De seguida, avançamos para o setup do segundo cliente a receber a *stream*. Começamos por construir uma página HTML (*video.html*), tal como referido no enunciado:

```
core@xubuncore:~$ vi video.html
core@xubuncore:~$ ls
core  Documents  Music  Pictures  Templates  videoB.mp4  Videos
Desktop  Downloads  ospf-mdr  Public  videoA.mp4  video.html
core@xubuncore:~$ cat video.html
<!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">
      A tag VIDEO não é suportada
    </video>
  </body>
</html>
core@xubuncore:~$
```

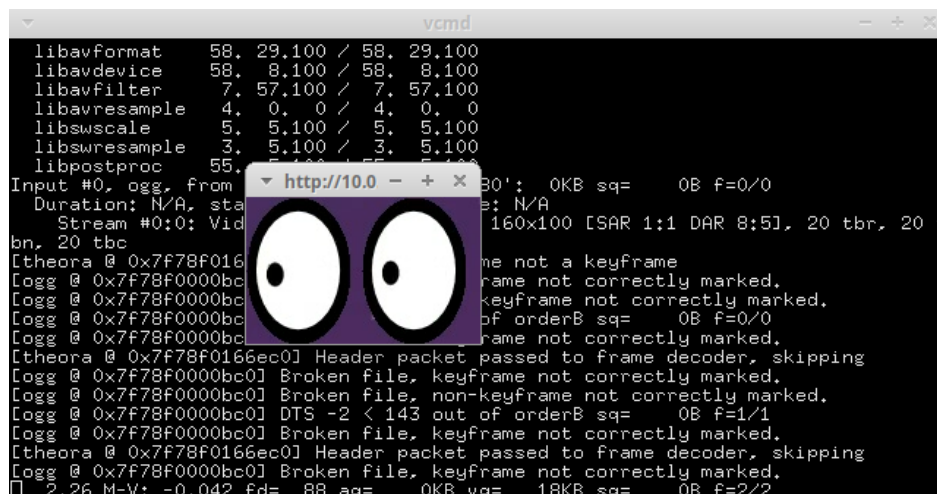
No portátil *Bela* executamos o *browser Firefox* abrindo a página criada anteriormente. Verificamos o funcionamento da segunda *stream* e procedemos à recolha de outra amostra de tráfego pelo *Wireshark*:



Streaming ESR: Etapa 1



Por fim, no portátil *Monstro*, usando o comando *ffplay*, criamos um terceiro cliente da *stream* de vídeo, efetuando uma terceira captura de tráfego:



Questão 1

Capture três pequenas amostras de tráfego no *link* de saída do servidor, respectivamente 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).

Começamos a nossa análise por verificar a largura de banda necessária para a transmissão de vídeo (taxa *bps*). Para isso, utilizamos o comando ***ffprobe videoA.mp4*** e observamos que, para este vídeo em particular, será necessária uma taxa de *bitrate* de 15kb/s.

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoA.mp4':
  Metadata:
    major_brand      : isom
    minor_version    : 512
    compatible_brands: isomiso2avc1mp41
    encoder          : Lavf58.29.100
  Duration: 00:00:07.20, start: 0.000000, bitrate: 15 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 160x100,
    12 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
    Metadata:
      handler_name    : VideoHandler
core@xubuncore:~$
```

Figura 4: Informação do vídeo utilizando o comando.

De seguida, analisamos um dos *packets* utilizados na transmissão da *stream*, de forma a perceber o encapsulamento utilizado. Neste *packet*, observamos que foi utilizado o protocolo *Ethernet* na camada de ligação lógica e o protocolo *IPv4* na camada de rede. Para além destes, nas camadas de transporte e aplicacional foram utilizados os protocolos *TCP* e *HTTP*, respetivamente. Também podemos observar que, dos 1514 *bytes* transmitidos, apenas 1233 *bytes* destes é que foram utilizados na transmissão efetiva da *stream*, sendo os restantes bytes utilizados pelas várias camadas protocolares na rede.

Taxa de encapsulamento: $(1233/1514) * 100 = 81.44\%$

No.	Time	Source	Destination	Protocol	Length	Info
26	0.465397744	10.0.0.20	10.0.0.10	TCP	66	32918 → 8080 [ACK]
27	1.459490500	10.0.0.10	10.0.0.20	TCP	1514	8080 → 32918 [ACK]
28	1.459491111	10.0.0.10	10.0.0.20	TCP	1514	8080 → 32918 [ACK]
<p>▶ Frame 27: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface veth5.0.fa, id 0</p> <p>▶ Ethernet II, Src: 00:00:00_aa:00:06 (00:00:00:aa:00:06), Dst: 00:00:00_aa:00:07 (00:00:00:aa:00:07)</p> <p>▶ Internet Protocol Version 4, Src: 10.0.0.10, Dst: 10.0.0.20</p> <p>▶ Transmission Control Protocol, Src Port: 8080, Dst Port: 32918, Seq: 13909, Ack: 1, Len: 1448</p> <p>▼ Hypertext Transfer Protocol</p> <p>▶ [truncated]OggS\000\000r\000\000\000\000\000\000\000\001\004\000\000\003\021\017\f\037\0Gdu?#\020\</p> <p>File Data: 1233 bytes</p> <p>▶ Data (1233 bytes)</p>						

Figura 5: Encapsulamento.

Passamos agora à análise dos vários fluxos gerados durante as *streams*, começando por observar o crescente impacto de tráfego na rede conforme o número de fluxos aumenta. Utilizando as funções estatísticas fornecidas pelo *Wireshark*, analisamos a estatística ”*Hierarchy*” de forma a observar um potencial crescimento de tráfego na rede presente no protocolo HTTP.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	368	100.0	241061	94 k	0	0	0
▼ Ethernet	100.0	368	2.1	5152	2022	0	0	0
▼ Internet Protocol Version 6	1.6	6	0.1	240	94	0	0	0
▼ User Datagram Protocol	0.5	2	0.0	16	6	0	0	0
Multicast Domain Name System	0.5	2	0.0	90	35	2	90	35
Open Shortest Path First	0.5	2	0.0	72	28	2	72	28
Internet Control Message Protocol v6	0.5	2	0.0	32	12	2	32	12
▼ Internet Protocol Version 4	97.8	360	3.0	7200	2825	0	0	0
▼ Transmission Control Protocol	95.1	350	94.5	227763	89 k	326	199698	78 k
▼ Hypertext Transfer Protocol	6.5	24	11.3	27324	10 k	21	26514	10 k
Malformed Packet	0.8	3	0.0	0	0	3	0	0
Open Shortest Path First	2.7	10	0.2	440	172	10	440	172
Address Resolution Protocol	0.5	2	0.0	56	21	2	56	21

Figura 6: Estatística *Hierarchy* com 1 *stream*.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	785	100.0	518622	199 k	0	0	0
▼ Ethernet	100.0	785	2.1	10990	4218	0	0	0
▼ Internet Protocol Version 6	0.3	2	0.0	80	30	0	0	0
Open Shortest Path First	0.3	2	0.0	72	27	2	72	27
▼ Internet Protocol Version 4	99.2	779	3.0	15580	5980	0	0	0
▼ Transmission Control Protocol	97.8	768	94.7	491304	188 k	708	422038	161 k
▼ Hypertext Transfer Protocol	7.6	60	13.0	67400	25 k	52	65240	25 k
Malformed Packet	1.0	8	0.0	0	0	8	0	0
Open Shortest Path First	1.4	11	0.1	484	185	11	484	185
Address Resolution Protocol	0.5	4	0.0	112	42	4	112	42

Figura 7: Estatística *Hierarchy* com 2 *streams*.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	1120	100.0	759801	291 k	0	0	0
▼ Ethernet	100.0	1120	2.1	15680	6005	0	0	0
▼ Internet Protocol Version 6	0.2	2	0.0	80	30	0	0	0
Open Shortest Path First	0.2	2	0.0	72	27	2	72	27
▼ Internet Protocol Version 4	99.5	1114	2.9	22280	8533	0	0	0
▼ Transmission Control Protocol	98.6	1104	94.9	721137	276 k	1038	645849	247 k
▼ Hypertext Transfer Protocol	5.9	66	9.6	73284	28 k	57	70854	27 k
Malformed Packet	0.8	9	0.0	0	0	9	0	0
Open Shortest Path First	0.9	10	0.1	440	168	10	440	168
Address Resolution Protocol	0.4	4	0.0	112	42	4	112	42

Figura 8: Estatística *Hierarchy* com 3 *streams*.

Como podemos observar, entre as três estatísticas, podemos concluir que a taxa de *bitrate* cresce conforme o número de *streams* aumenta, sendo estes valores **10kb/s**, **25kb/s** e **28kb/s**, respectivamente. Estes valores, apesar do crescimento observado ser o resultado esperado, é possível verificar algumas discrepâncias entre os valores teóricos de transmissão e os valores reais, como por exemplo, quando apenas há 1 *stream* presente, a taxa de tráfego é de apenas 10kb/s. No entanto, visto que existem flutuações na rede, consideramos estes valores adequados. Este comportamento de crescimento e discrepância também pode ser observado com o aumento de número de *streams*, sendo que o valor de taxa de *bitrate* aumenta progressivamente.

Em termos de análise de escalabilidade, podemos concluir que esta solução não é escalável, pois não só o servidor é um ponto único de falha, como também requer que cada cliente tenha um *link* de transmissão fixo com pelo menos 15kb/s de débito (como podemos verificar na figura 9) o que irá criar problemas de congestionamento de rede conforme o número de utilizadores aumenta. Por exemplo, quando este vídeo tiver que ser ”servido” a 100 ou até 10000 clientes, serão necessários *links* de 1.5mb/s ou 150mb/s, respetivamente, tomando em atenção que o vídeo utilizado neste exemplo contém uma taxa de transmissão exceccionalmente baixa.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	32918	10.0.0.10	8080	368	252 k	184	12 k	184	240 k	0.000000	20.8876	4651	92 k
10.0.2.20	36924	10.0.0.10	8080	368	252 k	184	12 k	184	240 k	0.000071	20.8876	4651	92 k
10.0.2.21	36430	10.0.0.10	8080	368	252 k	184	12 k	184	240 k	0.001266	20.8865	4651	92 k

Figura 9: Estatística *Conversations* com 3 *streams*.

2 *Streaming* adaptativo sobre HTTP (MPEG-DASH)

Tarefas

Para esta etapa, começamos por recodificar e produzir três versões do *videoB.mp4* de resoluções diferentes, obtendo estes resultados utilizando *ffprobe* nos vídeos gerados:

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoB_160_100_200k.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder          : Lavf58.29.100
Duration: 00:00:12.60, start: 0.000000, bitrate: 66 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 160x100, 63 kb/s, 30
fps, 30 tbr, 15360 tbn, 60 tbc (default)
Metadata:
  handler_name     : VideoHandler
core@xubuncore:~$
```

Figura 10: ffprobe videoB_160_100_200k.mp4

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoB_320_200_500k.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder          : Lavf58.29.100
Duration: 00:00:12.60, start: 0.000000, bitrate: 153 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 320x200, 150 kb/s, 30
fps, 30 tbr, 15360 tbn, 60 tbc (default)
Metadata:
  handler_name     : VideoHandler
core@xubuncore:~$
```

Figura 11: ffprobe videoB_320_200_500k.mp4

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoB_640_400_1000k.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder          : Lavf58.29.100
Duration: 00:00:12.60, start: 0.000000, bitrate: 381 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 640x400, 378 kb/s, 30
fps, 30 tbr, 15360 tbn, 60 tbc (default)
Metadata:
  handler_name     : VideoHandler
core@xubuncore:~$
```

Figura 12: ffprobe videoB_640_400_1000k.mp4

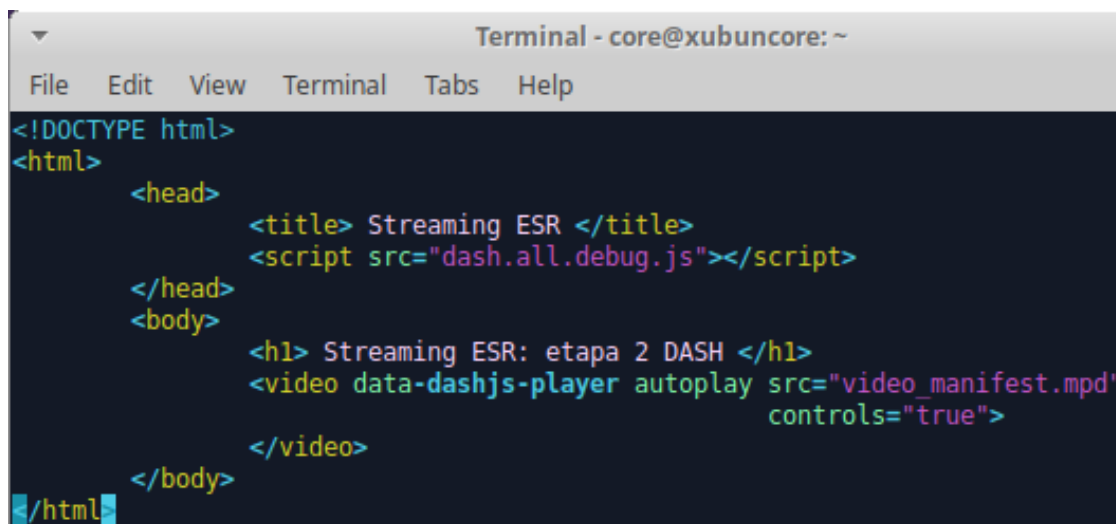
Podemos observar que, com o aumento da resolução do vídeo, a sua taxa de *bitrate* aumenta significativamente.

Produzimos, também, o ficheiro MPD com a descrição das alternativas, usando o MP4Box:

```
core@xubuncore:~$ MP4Box -dash 500 -out video_manifest videoB_160_100_200k.mp4 videoB_320_200_500k.mp4 videoB_640_400_1000k.mp4
DASH-ing files: 0.50s segments 0.50s fragments single sidx per segment
DASHing file videoB_160_100_200k.mp4
DASHing file videoB_320_200_500k.mp4
DASHing file videoB_640_400_1000k.mp4
[DASH] Generating MPD at time 2022-10-06T13:43:56.291Z
```

Figura 13: Ficheiro MPD gerado

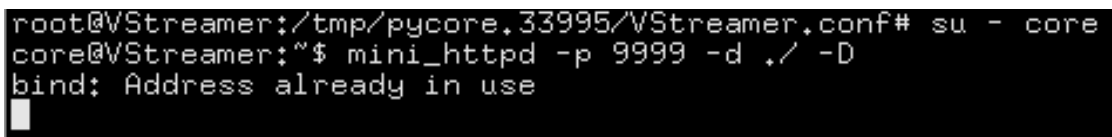
De seguida, construímos uma página HTML5 (*video_dash.html*) para visualizar o vídeo, que deverá incluir referências aos dois módulos *JavaScript* e referenciar na *tag* o vídeo *video_manifest.mpd*. Para isso, obtemos as *scripts* necessárias, através do comando *wget*.



```
Terminal - core@xubuncore: ~
File Edit View Terminal Tabs Help
<!DOCTYPE html>
<html>
  <head>
    <title> Streaming ESR </title>
    <script src="dash.all.debug.js"></script>
  </head>
  <body>
    <h1> Streaming ESR: etapa 2 DASH </h1>
    <video data-dashjs-player autoplay src="video_manifest.mpd"
          controls="true">
    </video>
  </body>
</html>
```

Figura 14: Ficheiro HTML5 criado

Com tudo preparado, testamos a conectividade na topologia e, depois de verificada, abrimos uma *bash* no servidor *VStreamer* e servimos o conteúdo com um servidor HTTP (*mini_httpd*):



```
root@VStreamer:/tmp/pycore.33995/VStreamer.conf# su - core
core@VStreamer:~$ mini_httpd -p 9999 -d ./ -D
bind: Address already in use
```

Figura 15: Servidor mini_httpd

Nos portáteis *Bela* e *Alladin*, abrimos o *Firefox* e visualizamos o conteúdo:

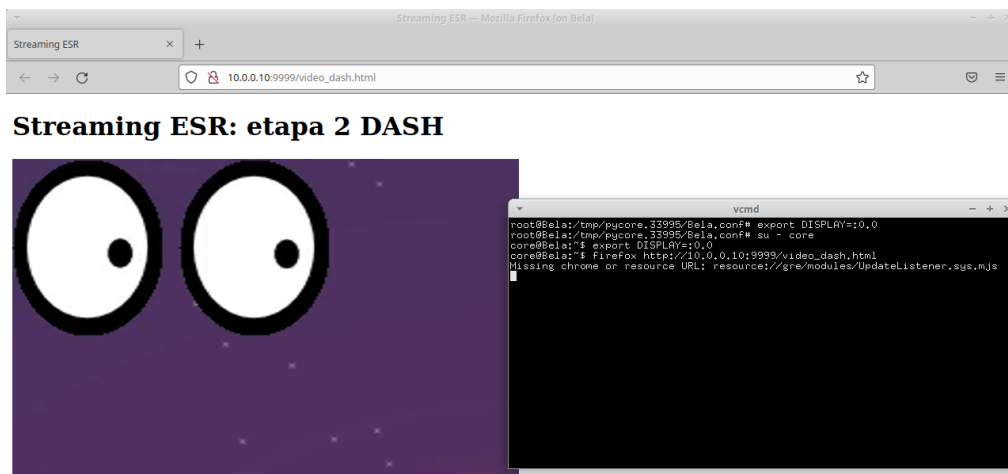


Figura 16: Stream no *browser* Bela

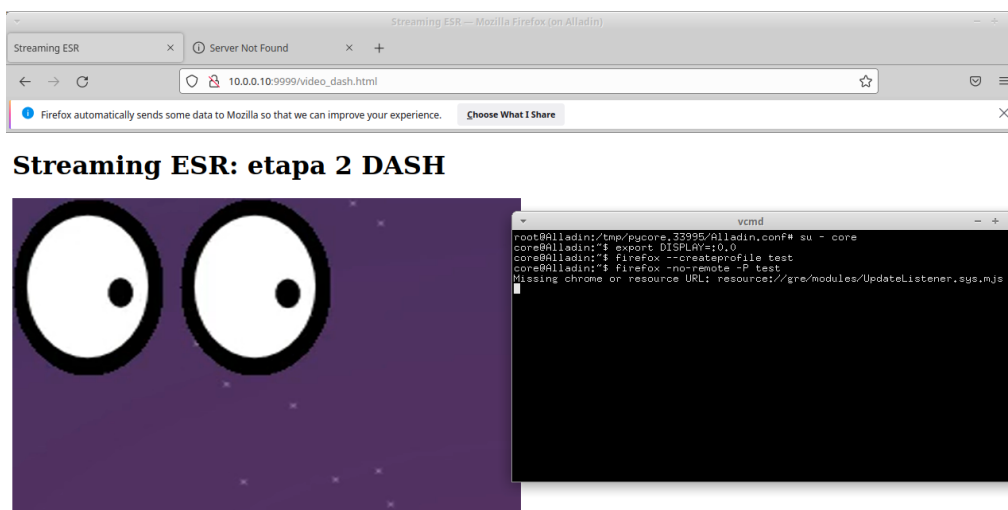


Figura 17: Stream no *browser* Alladin

Com as duas *streams* ativas, capturamos amostras de tráfego com o *Wireshark*.

Por fim, mexemos na capacidade do *link* entre o *sw2* e o portátil *Bela* de modo a forçar este a apresentar um vídeo de menor resolução. Decidimos limitar o *link* a 100kbs, visto que este limite encontra-se entre o vídeo de menor resolução e o seguinte, de 154kb/s e efetuamos uma nova captura de tráfego com esta limitação.

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.

Para a transmissão do *videoB* gerado, será necessário uma largura de banda mínima de 66kb/s (Figura 10), a qual corresponde à versão de vídeo de menor qualidade, e, para receber o vídeo de melhor qualidade, será necessário uma taxa de 381kb/s. (Figura 12).

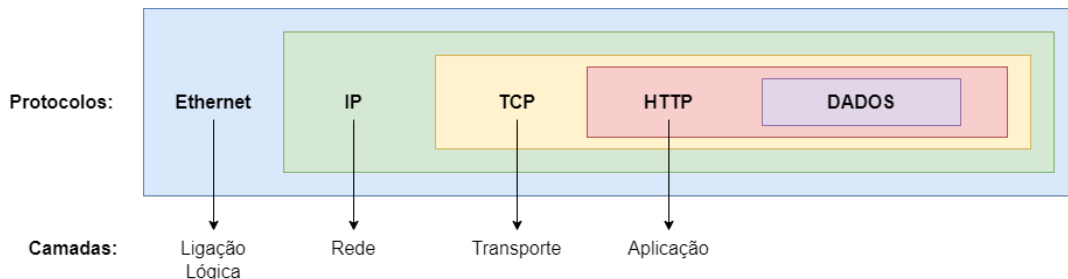
Relativamente à pilha protocolar, para conseguirmos observar a mesma, selecionamos uma trama capturada pelo *Wireshark* (figura 18), analisando o seu conteúdo:

No.	Time	Source	Destination	Protocol	Length	Info
43	25.517815245	10.0.0.21	10.0.0.10	TCP	66	39858 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0
44	25.518062429	10.0.0.21	10.0.0.10	HTTP	401	GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
45	25.518067573	10.0.0.10	10.0.0.21	TCP	66	9999 → 39858 [ACK] Seq=1 Ack=336 Win=64896 Len=0

Frame 44: 401 bytes on wire (3208 bits), 401 bytes captured (3208 bits) on interface veth5.0.4f, id 0
Ethernet II, Src: 00:00:00_aa:00:08 (00:00:00:aa:00:08), Dst: 00:00:00_aa:00:06 (00:00:00:aa:00:06)
Internet Protocol Version 4, Src: 10.0.0.21, Dst: 10.0.0.10
Transmission Control Protocol, Src Port: 39858, Dst Port: 9999, Seq: 1, Ack: 1, Len: 335
Hypertext Transfer Protocol
GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
Host: 10.0.0.10:9999
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:104.0) Gecko/20100101 Firefox/104.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Range: bytes=927-22261
Connection: keep-alive
Referer: http://10.0.0.10:9999/video_dash.html
[Full request URI: http://10.0.0.10:9999/videoB_640_400_1000k_dash.mp4]
[HTTP request 1/1]
[Response in frame: 753]

Figura 18: Camadas protocolares

Através da análise da trama, elaboramos um esquema onde simplificamos toda a informação contida na figura acima e demos destaque à ligação entre protocolos utilizados e camadas correspondentes para além do evidente encapsulamento entre as mesmas. Assim, obtivemos o seguinte esquema da pilha protocolar:



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 este exercício, como dito anteriormente, decidimos limitar o *link* de ligação do *sw2* e o portátil *Bela* para um débito de 100kb/s, de forma a forçar a mudança dinâmica do vídeo transmitido, visto que o vídeo de maior qualidade necessita de uma *bitrate* de 381kb/s, o de qualidade intermédia 153kb/s e o de menor resolução necessita apenas 66kb/s.

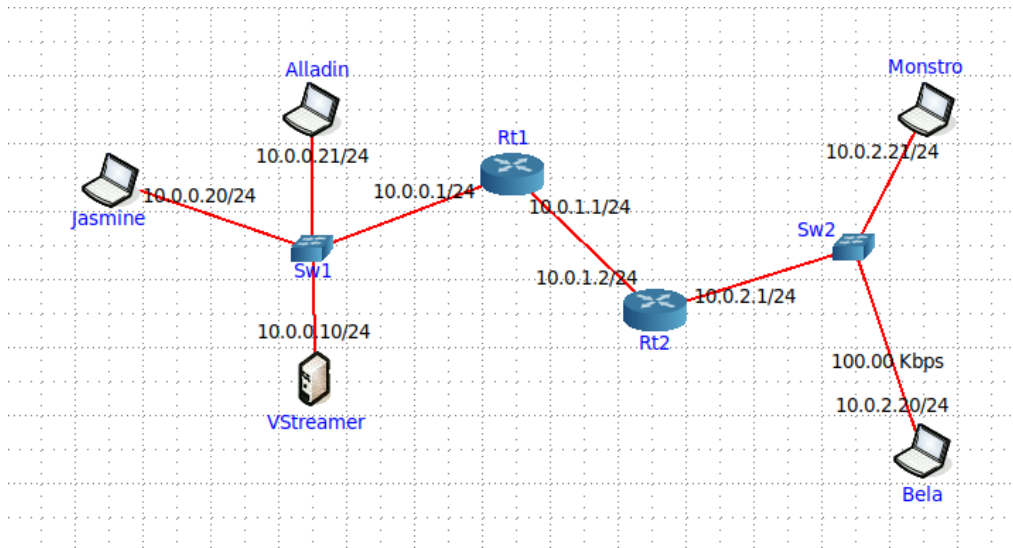


Figura 19: Topologia com link limitado

De seguida, voltamos a correr as streams em ambos os portáteis, assim como uma captura de tráfego do servidor *VStreamer*, obtendo as seguintes capturas relevantes:

No.	Time	Source	Destination	Protocol	Length	Info
16	12.375761933	10.0.2.20	10.0.0.10	TCP	74	42662 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=14601552...
17	12.375771749	10.0.0.10	10.0.2.20	TCP	74	9999 → 42662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 Tsv...
18	12.396360643	10.0.2.20	10.0.0.10	TCP	66	42662 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1460155321 TSecr=172048...
19	12.396852194	10.0.2.20	10.0.0.10	HTTP	381	GET /favicon.ico HTTP/1.1
20	12.396858067	10.0.0.10	10.0.2.20	TCP	66	9999 → 42662 [ACK] Seq=1 Ack=316 Win=64896 Len=0 TSval=1720485324 TSecr=1460...
21	12.396955481	10.0.0.10	10.0.2.20	HTTP	741	HTTP/1.1 404 Not Found (text/html)
22	12.472607528	10.0.2.20	10.0.0.10	TCP	66	42662 → 9999 [ACK] Seq=316 Ack=677 Win=64128 Len=0 TSval=1460155397 TSecr=17...
23	12.473330769	10.0.2.20	10.0.0.10	TCP	66	42662 → 9999 [FIN, ACK] Seq=316 Ack=677 Win=64128 Len=0 TSval=1460155398 TSe...
24	12.473336315	10.0.0.10	10.0.2.20	TCP	66	9999 → 42662 [ACK] Seq=677 Ack=317 Win=64896 Len=0 TSval=1720485401 TSecr=14...
25	12.789246588	10.0.2.20	10.0.0.10	TCP	74	42670 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=14601557...
26	12.789258165	10.0.0.10	10.0.2.20	TCP	74	9999 → 42670 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 Tsv...
27	12.799661381	10.0.2.20	10.0.0.10	TCP	66	42670 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1460155724 TSecr=172048...
28	12.799913855	10.0.2.20	10.0.0.10	HTTP	401	GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
29	12.799919473	10.0.0.10	10.0.2.20	TCP	66	9999 → 42670 [ACK] Seq=1 Ack=336 Win=64896 Len=0 TSval=1720485727 TSecr=1460...
30	12.800412129	10.0.0.10	10.0.2.20	TCP	1514	9999 → 42670 [ACK] Seq=1 Ack=336 Win=64896 Len=1448 TSval=1720485728 TSecr=1...

Figura 20: Captura Pedido GET no portátil Bela

No.	Time	Source	Destination	Protocol	Length	Info
106	15.958635446	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42674 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
107	15.961022347	10.0.2.20	10.0.0.10	TCP	74	[TCP Retransmission] 42674 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK...
108	15.961037115	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
109	15.961036210	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42674 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
110	16.023168174	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
111	16.079331907	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
112	16.198811472	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
113	16.214634851	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42682 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
114	16.216022424	10.0.2.20	10.0.0.10	TCP	74	[TCP Retransmission] 42682 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK...
115	16.216032532	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42682 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
116	16.328989755	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
117	16.456020728	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
118	16.572528719	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
119	16.688004075	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
120	16.813412984	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
121	16.932867169	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
122	17.055703008	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
123	17.175738195	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
124	17.296914157	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
125	17.414441789	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
126	17.535342282	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
127	17.663281797	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
128	17.783568287	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
129	17.898532615	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
130	17.971691507	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42674 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
131	17.971741949	10.0.2.20	10.0.0.10	TCP	74	[TCP Retransmission] 42674 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK...
132	17.971746392	10.0.0.10	10.0.2.20	TCP	74	[TCP Retransmission] 9999 → 42674 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS...
133	18.020605798	10.0.2.20	10.0.0.10	TCP	54	42670 → 9999 [RST] Seq=336 Win=0 Len=0
134	18.024206210	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
135	18.025614454	10.0.2.20	10.0.0.10	TCP	66	42674 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1460160950 TSecr=172048...
136	18.026330133	10.0.2.20	10.0.0.10	HTTP	402	GET /videoB_320_200_500k_dash.mp4 HTTP/1.1
137	18.026346561	10.0.0.10	10.0.2.20	TCP	66	9999 → 42674 [ACK] Seq=1 Ack=337 Win=64896 Len=0 TSval=1720490954 TSecr=1460...

Figura 21: Captura Erros e redução de qualidade no portátil Bela

No.	Time	Source	Destination	Protocol	Length	Info
199	20.118349388	00:00:00_aa:00:08	Broadcast	ARP	42	Who has 10.0.0.10? Tell 10.0.0.21
200	20.118364352	00:00:00_aa:00:08	00:00:00_aa:00:08	ARP	42	10.0.0.10 is at 00:00:00_aa:00:08
201	20.118373706	10.0.0.21	10.0.0.10	TCP	74	43866 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=25741032...
202	20.118381609	10.0.0.10	10.0.0.21	TCP	74	9999 → 43866 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSv...
203	20.118390212	10.0.0.21	10.0.0.10	TCP	66	43866 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2574103272 TSecr=172107...
204	20.118667644	10.0.0.21	10.0.0.10	HTTP	381	GET /favicon.ico HTTP/1.1
205	20.118672854	10.0.0.10	10.0.0.21	TCP	66	9999 → 43866 [ACK] Seq=1 Ack=316 Win=64896 Len=0 TSval=1721075834 TSecr=2574...
206	20.118754652	10.0.0.10	10.0.0.21	HTTP	741	HTTP/1.1 404 Not Found (text/html)
207	20.118801758	10.0.0.21	10.0.0.10	TCP	66	43866 → 9999 [ACK] Seq=316 Ack=677 Win=64128 Len=0 TSval=2574103272 TSecr=17...
208	20.118857950	10.0.0.21	10.0.0.10	TCP	66	43866 → 9999 [FIN, ACK] Seq=316 Ack=677 Win=64128 Len=0 TSval=2574103272 TSe...
209	20.118860254	10.0.0.10	10.0.0.21	TCP	66	9999 → 43866 [ACK] Seq=677 Ack=317 Win=64896 Len=0 TSval=1721075834 TSecr=25...
210	20.213466832	10.0.2.20	10.0.0.10	TCP	66	42674 → 9999 [ACK] Seq=337 Ack=24617 Win=64256 Len=0 TSval=1460163137 TSecr=...
211	20.213482111	10.0.0.10	10.0.2.20	TCP	1514	9999 → 42674 [ACK] Seq=57921 Ack=337 Win=64896 Len=1448 TSval=1720493141 TSe...
212	20.378956301	10.0.2.20	10.0.0.10	TCP	66	42674 → 9999 [ACK] Seq=337 Ack=26065 Win=64256 Len=0 TSval=1460163302 TSecr=...
213	20.378982063	10.0.0.10	10.0.2.20	TCP	1514	9999 → 42674 [ACK] Seq=59369 Ack=337 Win=64896 Len=1448 TSval=1720493306 TSe...
214	20.448918354	10.0.0.21	10.0.0.10	TCP	74	43870 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=25741036...
215	20.448933186	10.0.0.10	10.0.0.21	TCP	74	9999 → 43870 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSv...
216	20.448941979	10.0.0.21	10.0.0.10	TCP	66	43870 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2574103602 TSecr=172107...
217	20.449209844	10.0.0.21	10.0.0.10	HTTP	401	GET /videoB_640_400_1000k_dash.mp4 HTTP/1.1
218	20.449215188	10.0.0.10	10.0.0.21	TCP	66	9999 → 43870 [ACK] Seq=1 Ack=336 Win=64896 Len=0 TSval=1721076165 TSecr=2574...
219	20.449327373	10.0.0.10	10.0.0.21	TCP	1514	9999 → 43870 [ACK] Seq=1 Ack=336 Win=64896 Len=1448 TSval=1721076165 TSecr=2...

Figura 22: Captura Pedido GET no portátil Alladin

No.	Time	Source	Destination	Protocol	Length	Info
959	37.969940127	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
960	38.041852775	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
961	38.091018857	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
962	38.222605074	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
963	38.333291984	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
964	38.455524139	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
965	38.575450330	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
966	38.696632294	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
967	38.805545595	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=337 Win=0 Len=0
968	38.830936848	10.0.2.20	10.0.0.10	TCP	54	42674 → 9999 [RST] Seq=338 Win=0 Len=0
969	38.836498019	10.0.2.20	10.0.0.10	TCP	66	51978 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1460181761 TSecr=172050...
970	38.836871439	10.0.2.20	10.0.0.10	HTTP	402	GET /videoB_320_200_500k_dash.mp4 HTTP/1.1
971	38.836878949	10.0.0.10	10.0.2.20	TCP	66	9999 → 51978 [ACK] Seq=1 Ack=337 Win=64896 Len=0 TSval=1720511764 TSecr=1460...
972	38.837443315	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=1 Ack=337 Win=64896 Len=1448 TSval=1720511765 TSecr=1...
973	38.837443604	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=1449 Ack=337 Win=64896 Len=1448 TSval=1720511765...
974	38.837461258	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=2897 Ack=337 Win=64896 Len=1448 TSval=1720511765 TSec...
975	38.837461405	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=4345 Ack=337 Win=64896 Len=1448 TSval=1720511765...
976	38.837471366	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=5793 Ack=337 Win=64896 Len=1448 TSval=1720511765 TSec...
977	38.837471478	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=7241 Ack=337 Win=64896 Len=1448 TSval=1720511765...
978	38.837481153	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=8689 Ack=337 Win=64896 Len=1448 TSval=1720511765 TSec...
979	38.837481266	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=10137 Ack=337 Win=64896 Len=1448 TSval=172051176...
980	38.837491397	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=11585 Ack=337 Win=64896 Len=1448 TSval=1720511765 TSe...
981	38.837491509	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=13033 Ack=337 Win=64896 Len=1448 TSval=172051176...
982	38.848668068	10.0.2.20	10.0.0.10	TCP	66	51982 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1460181773 TSecr=172050...
983	38.854432004	10.0.2.20	10.0.0.10	TCP	66	[TCP Dup ACK 982#1] 51978 → 9999 [ACK] Seq=337 Ack=1 Win=64256 Len=0 TSval=1...
984	38.866324599	10.0.2.20	10.0.0.10	TCP	66	[TCP Dup ACK 982#1] 51982 → 9999 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=146...
985	39.006275797	10.0.2.20	10.0.0.10	TCP	66	51978 → 9999 [ACK] Seq=337 Ack=1449 Win=62048 Len=0 TSval=1460181931 TSecr=1...
986	39.006309994	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [ACK] Seq=14481 Ack=337 Win=64896 Len=1448 TSval=1720511934 TSe...
987	39.006310377	10.0.0.10	10.0.2.20	TCP	1514	9999 → 51978 [PSH, ACK] Seq=15929 Ack=337 Win=64896 Len=1448 TSval=172051193...
988	39.013283143	10.0.2.20	10.0.0.10	TCP	66	51978 → 9999 [FIN, ACK] Seq=337 Ack=1449 Win=64128 Len=0 TSval=1460181938 TS...
989	39.037115957	10.0.2.20	10.0.0.10	HTTP	400	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
990	39.037130685	10.0.0.10	10.0.2.20	TCP	66	9999 → 51982 [ACK] Seq=1 Ack=335 Win=64896 Len=0 TSval=1720511965 TSecr=1460...

Figura 23: Captura Erros e redução de qualidade no portátil Bela

Nas capturas seguintes salientamos os seguintes pontos:

- **Figura 19 - Packet Nº28:** O portátil Bela efetua um pedido GET do vídeo de maior qualidade possível.
- **Figura 20 - Packet Nº136:** Após vários erros de transmissão e retransmissões, o portátil Bela opta por diminuir a resolução do vídeo, efetuando um pedido GET do vídeo de qualidade intermédia.
- **Figura 21 - Packet Nº217:** O portátil Alladin efetua um pedido GET do vídeo de maior qualidade possível. Esta transmissão ocorre sem nenhum problema.
- **Figura 22 - Packet Nº970:** Após mais alguns erros de transmissão, o portátil Bela tenta retransmitir o vídeo da mesma qualidade.
- **Figura 22 - Packet Nº989:** Com mais alguns erros, o portátil Bela acaba por pedir a transmissão do vídeo de menor qualidade, de forma a suportar a taxa de débito disponível na sua rede.

Questão 4

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

No DASH, o ficheiro de vídeo é codificado em versões diferentes, com cada versão a ter uma taxa de *bits* diferente e, correspondentemente, um nível de qualidade diferente. O cliente solicita dinamicamente *chunks* de vídeo de alguns segundos de duração através de pedidos HTTP GET. Quando a quantidade de largura de banda disponível é alta, o cliente seleciona *chunks* de uma versão de alta qualidade e, naturalmente, quando a largura de banda disponível é baixa, seleciona a partir de uma versão de baixa qualidade.

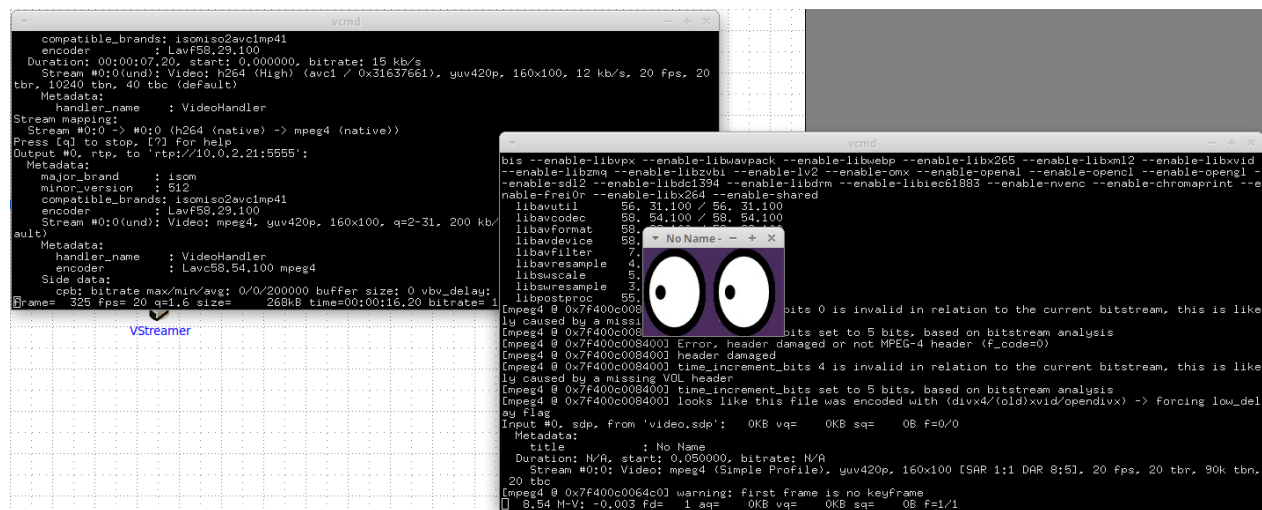
Assim, o papel do protocolo DASH é permitir a alteração dinâmica do vídeo transmitido pela rede, de modo a diminuir ou aumentar a taxa de *bitrate* necessária para a sua transmissão, permitindo ao cliente alternar livremente entre diferentes níveis de qualidade. Como vimos no exercício anterior, por exemplo, quando o portátil Bela encontrou a sua rede sobrecarregada com a transmissão da *stream*, este conseguiu trocar o vídeo para um semelhante, mas de menor qualidade de forma a facilitar a sua transmissão.

Para além disto, cada versão de vídeo é armazenada no servidor HTTP, cada uma com um URL diferente. O servidor HTTP contém um ficheiro MPD que fornece um URL para cada versão, juntamente com a sua taxa de *bits* - **manifesto**. Assim, o cliente solicita primeiro este ficheiro de manifesto, tomando conhecimento das várias versões disponíveis.

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

Tarefas

Começamos por utilizar e testar a conectividade na topologia criada na etapa 1: (Figura 1 e Figura 2). De seguida, no servidor *VStreamer*, iniciamos uma sessão de *streaming* com RTP através do *ffmpeg* e no cliente *Monstro* iniciamos um cliente a receber a *stream*. Também capturamos o tráfego com o *Wireshark* no *link* de saída do servidor.



Por fim, passamos para o exercício de *multicast*, ou seja, desenvolvemos uma topologia com apenas um *switch*, um servidor e quatro portáteis como indicado e testamos a sua conectividade:

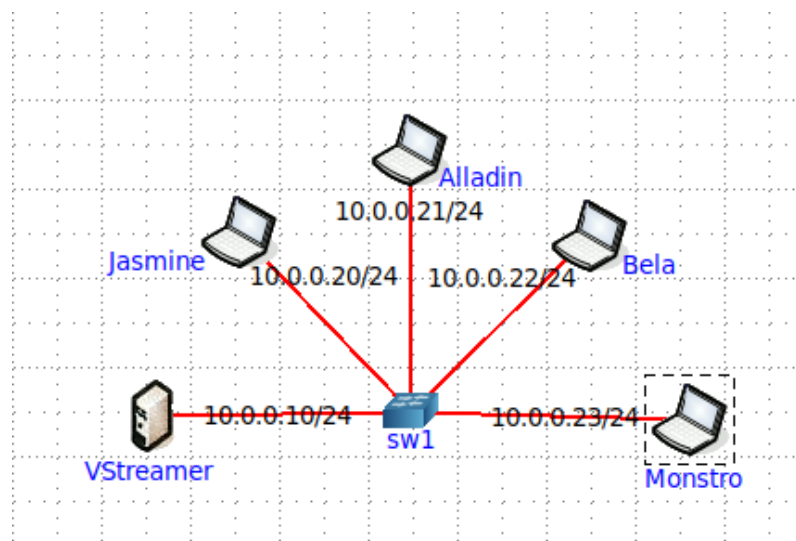


Figura 24: Topologia Multicast


```

root@Jasmine:/tmp/pycore.37531/Jasmine.conf# ping 10.0.0.23
PING 10.0.0.23 (10.0.0.23) 56(84) bytes of data:
64 bytes from 10.0.0.23: icmp_seq=1 ttl=64 time=0.117 ms
64 bytes from 10.0.0.23: icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 10.0.0.23: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.0.0.23: icmp_seq=4 ttl=64 time=0.076 ms
^C
--- 10.0.0.23 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3032ms
rtt min/avg/max/mdev = 0.066/0.085/0.117/0.019 ms
root@Jasmine:/tmp/pycore.37531/Jasmine.conf#

```

Figura 25: Ping Jasmine → Monstro

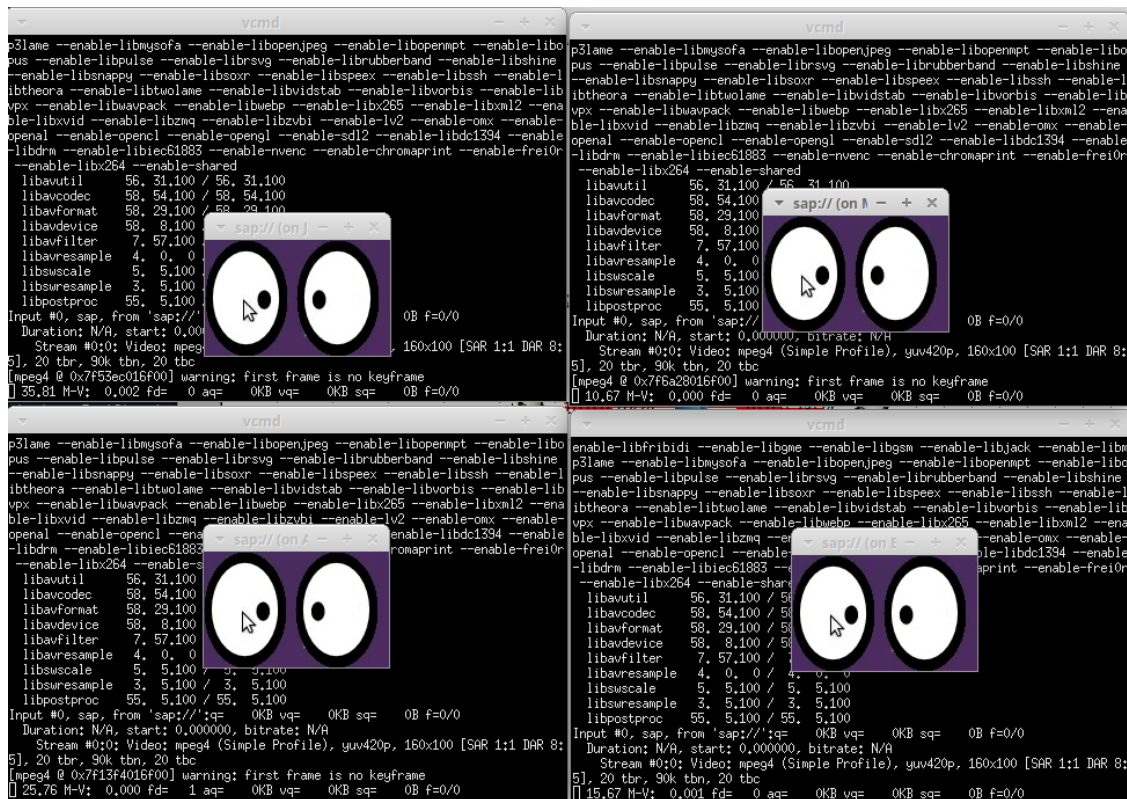
No servidor *VStreamer*, iniciamos uma sessão de *streaming multicast* com o *ffmpeg*:

```

Stream mapping:
  Stream #0:0 -> #0:0 (h264 (native) -> mpeg4 (native))
Press [q] to stop, [?] for help
Output #0, sap, to 'sap://224.0.0.200:5555':
  Metadata:
    major_brand      : isom
    minor_version    : 512
    compatible_brands: isomiso2avc1mp41
    encoder          : Lavf58.29.100
  Stream #0:0(und): Video: mpeg4, yuv420p, 160x100, q=2-31, 200 kb/s, 20 fps, 90k tbn, 20 tbc (default)
  Metadata:
    handler_name     : VideoHandler
    encoder          : Lavc58.54.100 mpeg4
  Side data:
    cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
[Frame= 294 fps= 20 q=2.0 size=N/A time=00:00:14.65 bitrate=N/A speed= 1x

```

E, de seguida, em cada um dos portáteis (Jasmine, Aladdin, Bela e Monstro), efetuamos a conexão à *stream* finalizando com uma captura do tráfego de saída no servidor *VStreamer*:



Questão 5

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

Começamos por apresentar as várias estatísticas recolhidas nos cenários *Unicast* e *Multicast*:

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	577	100.0	376455	144 k	0	0	0
▼ Ethernet	100.0	577	2.1	8078	3106	0	0	0
▼ Internet Protocol Version 6	0.3	2	0.0	80	30	0	0	0
Open Shortest Path First	0.3	2	0.0	72	27	2	72	27
▼ Internet Protocol Version 4	99.7	575	3.1	11500	4422	0	0	0
▼ User Datagram Protocol	97.7	564	1.2	4512	1735	0	0	0
Data	97.2	561	93.1	350433	134 k	561	350433	134 k
ADwin configuration protocol	0.5	3	0.3	1296	498	3	1296	498
Open Shortest Path First	1.9	11	0.1	484	186	11	484	186

Figura 26: Estatística *Hierarchy* no cenário *Unicast*

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.10	38247	10.0.2.21	5555	560	375 k	560	375 k	0	0	0.000000	20.8044	144 k	0
10.0.0.10	38248	10.0.2.21	5556	4	280	4	280	0	0	3.601622	15.0502	148	0

Figura 27: Estatística *Conversations* no cenário *Unicast*

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	562	100.0	365933	142 k	0	0	0
▼ Ethernet	100.0	562	2.2	7868	3054	0	0	0
▼ Internet Protocol Version 6	0.4	2	0.0	80	31	0	0	0
Internet Control Message Protocol v6	0.4	2	0.0	32	12	2	32	12
▼ Internet Protocol Version 4	99.6	560	3.1	11200	4348	0	0	0
▼ User Datagram Protocol	99.6	560	1.2	4480	1739	0	0	0
Session Announcement Protocol	0.7	4	0.4	1296	503	0	0	0
Session Description Protocol	0.7	4	0.3	1200	465	4	1200	465
▼ Real-Time Transport Protocol	84.0	472	77.4	283239	109 k	0	0	0
MP4V-ES	84.0	472	75.9	277575	107 k	472	277575	107 k
Real-time Transport Control Protocol	0.7	4	0.0	112	43	4	112	43
Data	14.1	79	15.6	57194	22 k	79	57194	22 k
ADwin configuration protocol	0.2	1	0.1	432	167	1	432	167

Figura 28: Estatística *Hierarchy* no cenário *Multicast*

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.10	56833	224.0.0.200	5555	552	364 k	552	364 k	0	0	0.000000	20.6041	141 k	0
10.0.0.10	40702	224.2.127.254	9875	4	1464	4	1464	0	0	3.008654	15.0587	777	0
10.0.0.10	56834	224.0.0.200	5556	4	280	4	280	0	0	3.008730	15.0586	148	0

Figura 29: Estatística *Conversations* no cenário *Multicast*

No cenário *unicast*, podemos verificar que temos apenas uma conversação, correspondente à *stream* a ser transmitida, que ocupa uma largura de banda de 134kb/s.

Já no cenário *multicast*, encontramos dados semelhantes, havendo apenas uma conversação correspondente à *stream*, que ocupa uma largura de banda semelhante de 107kb/s. No entanto, é importante notar que neste cenário, esta *stream* está a ser visualizada por 4 clientes em simultâneo, contrariamente ao cenário de *unicast* que apenas está a servir um cliente.

Este fator indica-nos que, no cenário *multicast*, não há a necessidade de transmitir múltiplas *streams* para cada cliente ligado, sendo estes capazes de "partilhar" a mesma transmissão através do protocolo *multicast* que permite a replicação da mesma pelos nodos da rede. Esta diferença é vantajosa pois resolve o problema de escalabilidade presente nos exercícios

anteriores, pois independentemente do número de clientes ligados à *stream*, o tráfego na rede não aumenta.

No entanto, a solução *multicast* traz algumas pequenas desvantagens, pois é necessária a gestão dos vários grupos *multicast* presentes na rede, assim como o tráfego extra proveniente dos pacotes de "anúncio" (*flooding*) que são utilizados, apesar destes não terem um peso significativo na rede em relação aos ganhos que a solução permite.

Conclusão

Com o terminar desta trabalho prático, encontramos-nos satisfeitos com o trabalho desenvolvido, tendo alcançado todas as metas propostas pelos docentes, assim como ter justificado adequadamente os resultados observados e analisados.

Este trabalho também nos permitiu aprofundar o nosso conhecimento na área de engenharia de serviços em rede através da aplicação prática dos conteúdos lecionados nas aulas teóricas. Desenvolvemos uma melhor perceção acerca do funcionamento dos protocolos de *streaming* de vídeo assim como também pudemos observar e compreender de forma mais clara as soluções *Unicast* e *Multicast* e as diferenças entre estas.