

Trabalho Prático Nº1 – Streaming de áudio e vídeo a pedido e em tempo real

André Lucena Ribas Ferreira pg52672
Carlos Eduardo da Silva Machado pg52675
Gonçalo Manuel Maia de Sousa pg52682

12 de outubro de 2023

Questões e Respostas

Etapa 1

Topologia Core:

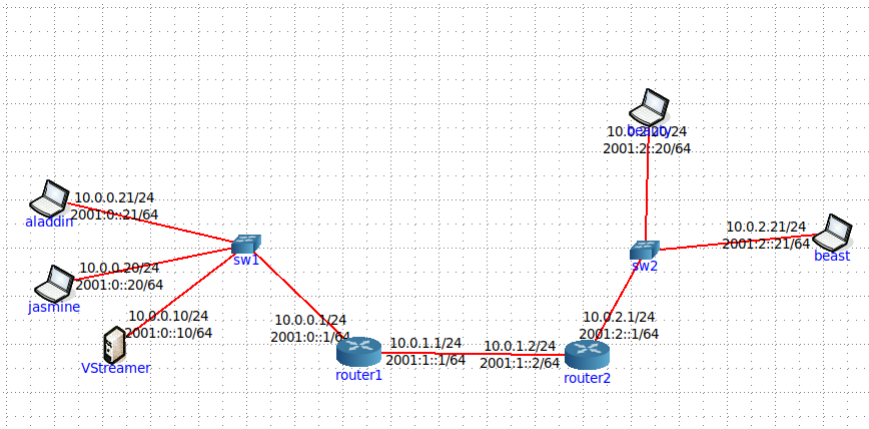


Figura 1: Topologia *core* utilizada na Etapa 1.

Imagens dos vídeos a correr em simultâneo:

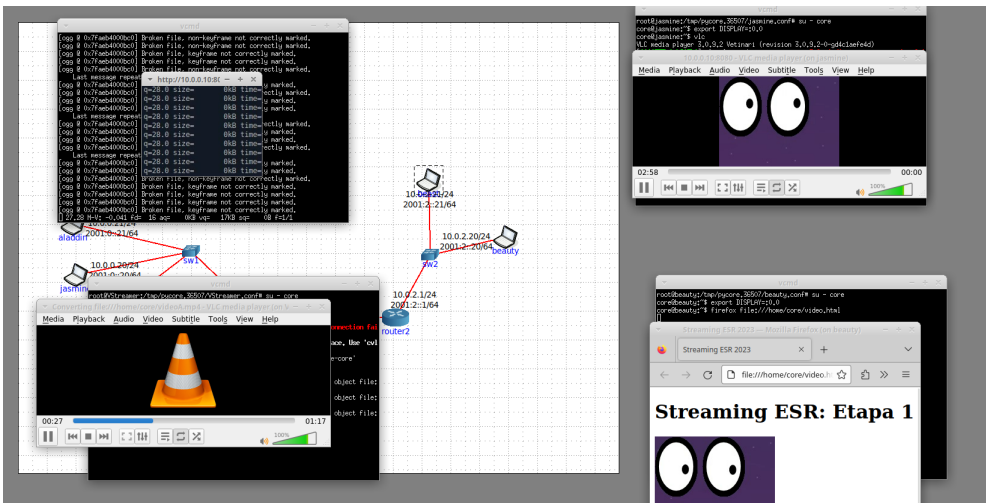


Figura 2: Vídeos a correr em simultâneo nos 3 utilizadores.

Questão 1: Capture três pequenas amostras de tráfego 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 bit/s 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).

O videoA tem necessidade de 25 kbit/s de taxa, segundo o comando *ffmpeg*:

```
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 --enable-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libchromaprint --enable-libcodec2 --enable-libcoke --enable-libcopenhls --enable-libcopenrtmp --enable-libcpl --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-liblensfun --enable-liblsmash --enable-libmfx --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librav1e --enable-librist --enable-librubio --enable-libSDL2 --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libsvt-av1 --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2 --enable-libyuv --enable-libz --enable-libzimg --enable-libzmq --enable-libzvbi --enable-lto --enable-vcpkg --enable-zlib
  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          : Lavf58.29.100
  Duration: 00:01:17.40, start: 0.000000, bitrate: 27 kb/s
  Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 200x150, 25 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
  Metadata:
    handler name     : VideoHandler
```

Figura 3: Informação do *ffmpeg* sobre o ficheiro videoA.mp4.

O encapsulamento utilizada neste contexto é a seguinte:

- Camada de ligação (*Link Layer*) - Ethernet II
- Camada de rede (*Network Layer*) - IP (*Internet Protocol*)
- Camada de transporte (*Transport Layer*) - TCP (*Transmission Control Protocol*)

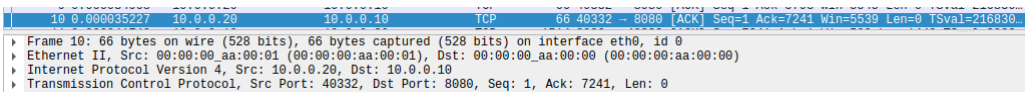


Figura 4: Encapsulamento de um pacote enviado.

Para **1 cliente** (VLC) nota-se uma taxa de 26 kbit/s, ou seja, um valor quase idêntico ao emitido pelo *ffmpeg* sobre o videoA:

O número total de fluxos é 1:

Wireshark - Conversations - pl21_tp1_ex1_single.pcapng												
Ethernet 3		IPv4 2		IPv6 1		TCP 1		UDP				
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
10.0.0.20	40332	10.0.0.10	8080	382	270 k	191	12 k	191	257 k	0.000000	14.7520	6836

Figura 6: Comunicações TCP a partir do VStreamer com **1 cliente**.

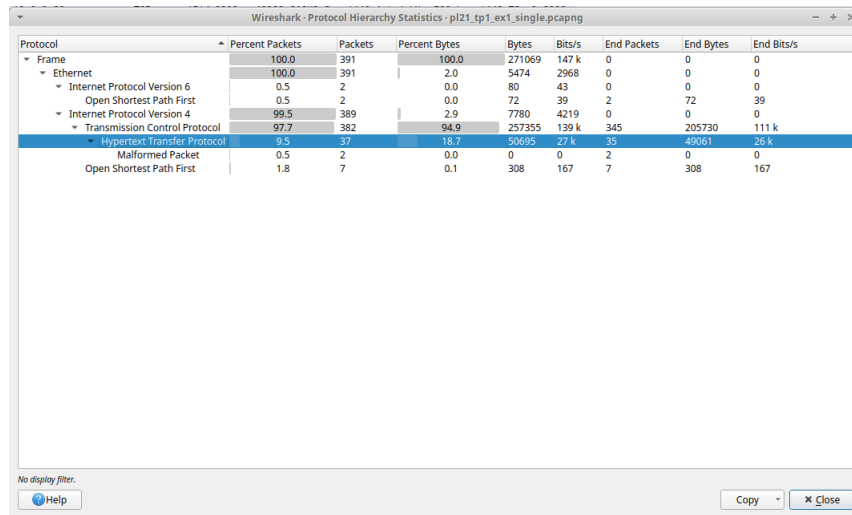


Figura 5: Estatísticas da captura *Wireshark* sobre o VStreamer com **1 cliente**.

Para **2 clientes** (VLC e Firefox) nota-se uma taxa de 38 kbit/s segundo o *Wireshark*:

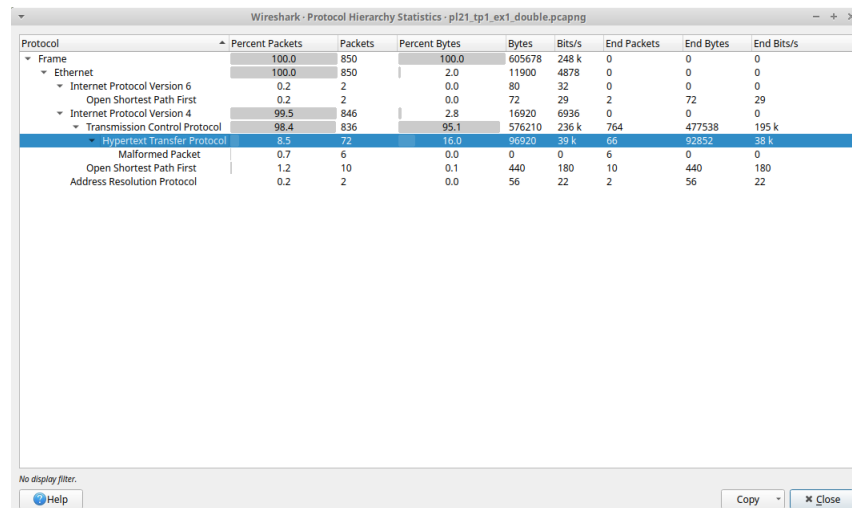


Figura 7: Estatísticas da captura *Wireshark* sobre o VStreamer com **2 clientes**.

O número total de fluxos é 2:

Wireshark - Conversations - pl21_tp1_ex1_double.pcapng													
Ethernet · 4		IPv4 · 3		IPv6 · 1		TCP · 2		UDP					
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	40332	10.0.0.10	8080	418	302 k	209	13 k	209	288 k	0.000000	19.5134	5655	
10.0.2.20	60856	10.0.0.10	8080	418	302 k	209	13 k	209	288 k	0.000062	19.5134	5655	

Figura 8: Comunicações TCP a partir do VStreamer com **2 clientes**.

Para **3 clientes** (VLC, Firefox e ffmpeg) nota-se uma taxa de 57 kbit/s segundo o *Wireshark*:

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	1033	100.0	739131	329 k	0	0	0
Ethernet	100.0	1033	2.0	14462	6439	0	0	0
Internet Protocol Version 6	0.2	2	0.0	80	35	0	0	0
Open Shortest Path First	0.2	2	0.0	72	32	2	72	32
Internet Protocol Version 4	99.6	1029	2.8	20580	9163	0	0	0
Transmission Control Protocol	98.7	1020	95.2	703485	313 k	927	570291	253 k
Hypertext Transfer Protocol	9.0	93	17.7	130815	58 k	90	129924	57 k
Malformed Packet	0.3	3	0.0	0	0	3	0	0
Open Shortest Path First	0.9	9	0.1	396	176	9	396	176
Address Resolution Protocol	0.2	2	0.0	56	24	2	56	24

Figura 9: Estatísticas da captura *Wireshark* sobre o VStreamer com **3** clientes.

O número total de fluxos é 3:

Wireshark - Conversations - pl21_tp1_ex1_triple.pcapng

Ethernet · 4		IPv4 · 4		IPv6 · 1		TCP · 3		UDP						
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	40332	10.0.0.10	8080	340	246 k	170	11 k	170	234 k	0.000000	17.9667	4995		
10.0.2.20	60866	10.0.0.10	8080	340	246 k	170	11 k	170	234 k	0.000025	17.9669	4995		
10.0.2.21	41354	10.0.0.10	8080	340	246 k	170	11 k	170	234 k	0.000018	17.9669	4995		

Figura 10: Comunicações TCP a partir do VStreamer sobre o VStreamer com **3** clientes.

Confirma-se um crescimento gradual da taxa necessária com o aumentar do número de utilizadores, o que representa um problema de escalabilidade, independentemente do formato de visualização.

Etapa 2

Nesta Etapa, a topologia foi a mesma da etapa anterior.

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.

Neste cenário, produziram-se 3 cópias do videoB com diferentes resoluções. Um vídeo com a resolução 200x150, outro com 480x360 e um último com 640x480.

Vídeo de resolução 200x150:

```
</SegmentList>
<Representation id="1" mimeType="video/mp4" codecs="avc3.64000c" width="200" height="150" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="113040">
  <BaseURL>videoB_200_150_200k_dash.mp4</BaseURL>
  <SegmentList timescale="15360" duration="7680">
    <SegmentURL mediaRange="928-5815" indexRange="928-971"/>
    <SegmentURL mediaRange="5816-10390" indexRange="5816-5859"/>
    <SegmentURL mediaRange="10391-15886" indexRange="10391-10434"/>
    <SegmentURL mediaRange="15887-25764" indexRange="15887-15930"/>
    <SegmentURL mediaRange="25765-31481" indexRange="25765-25808"/>
    <SegmentURL mediaRange="31482-37688" indexRange="31482-31525"/>
    <SegmentURL mediaRange="37689-49006" indexRange="37689-37732"/>
    <SegmentURL mediaRange="49007-55611" indexRange="49007-49050"/>
    <SegmentURL mediaRange="55612-63809" indexRange="55612-55655"/>
    <SegmentURL mediaRange="63810-70787" indexRange="63810-63853"/>
    <SegmentURL mediaRange="70788-84134" indexRange="70788-70831"/>
    <SegmentURL mediaRange="84135-92168" indexRange="84135-84178"/>
    <SegmentURL mediaRange="92169-100093" indexRange="92169-92212"/>
    <SegmentURL mediaRange="100094-114770" indexRange="100094-100137"/>
    <SegmentURL mediaRange="114771-119968" indexRange="114771-114814"/>
    <SegmentURL mediaRange="119969-127962" indexRange="119969-120012"/>
    <SegmentURL mediaRange="127963-143406" indexRange="127963-128006"/>
    <SegmentURL mediaRange="143407-148387" indexRange="143407-143450"/>
    <SegmentURL mediaRange="148388-155411" indexRange="148388-148431"/>
    <SegmentURL mediaRange="155412-160157" indexRange="155412-155455"/>
    <SegmentURL mediaRange="160158-172084" indexRange="160158-160201"/>
    <SegmentURL mediaRange="172085-177075" indexRange="172085-172128"/>
    <SegmentURL mediaRange="177076-182341" indexRange="177076-177119"/>
    <SegmentURL mediaRange="182342-191641" indexRange="182342-182385"/>
    <SegmentURL mediaRange="191642-193883" indexRange="191642-191685"/>
    <SegmentURL mediaRange="193884-197938" indexRange="193884-193927"/>
    <SegmentURL mediaRange="197939-208553" indexRange="197939-197982"/>
    <SegmentURL mediaRange="208554-212636" indexRange="208554-208597"/>
    <SegmentURL mediaRange="212637-216156" indexRange="212637-212680"/>
    <SegmentURL mediaRange="216157-219413" indexRange="216157-216200"/>
    <SegmentURL mediaRange="219414-229785" indexRange="219414-219457"/>
    <SegmentURL mediaRange="229786-230385" indexRange="229786-229829"/>
    <SegmentURL mediaRange="230386-230790" indexRange="230386-230429"/>
  </SegmentList>
</Representation>
```

Figura 11: *Bitrate* e segmentos do vídeo de resolução 200x150.

Como se pode observar na imagem acima, o menor vídeo tem um *bitrate* de 113040 bit/s.

Vídeo de resolução 480x360:

```
<Representation id="2" mimeType="video/mp4" codecs="avc3.64001e" width="480" height="360" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="316123">
<BaseURL>video8_480_360_500k_dash.mp4</BaseURL>
<SegmentList timescale="15360" duration="7680">
  <SegmentURL mediaRange="928-12196" indexRange="928-971"/>
  <SegmentURL mediaRange="12197-25377" indexRange="12197-12240"/>
  <SegmentURL mediaRange="25378-42142" indexRange="25378-25421"/>
  <SegmentURL mediaRange="42143-69487" indexRange="42143-42186"/>
  <SegmentURL mediaRange="69488-83672" indexRange="69488-69531"/>
  <SegmentURL mediaRange="83673-101109" indexRange="83673-83716"/>
  <SegmentURL mediaRange="101110-135165" indexRange="101110-101153"/>
  <SegmentURL mediaRange="135166-151513" indexRange="135166-135209"/>
  <SegmentURL mediaRange="151514-173895" indexRange="151514-151557"/>
  <SegmentURL mediaRange="173896-194906" indexRange="173896-173939"/>
  <SegmentURL mediaRange="194907-233956" indexRange="194907-194950"/>
  <SegmentURL mediaRange="233957-255297" indexRange="233957-234000"/>
  <SegmentURL mediaRange="255298-276649" indexRange="255298-255341"/>
  <SegmentURL mediaRange="276650-317638" indexRange="276650-276693"/>
  <SegmentURL mediaRange="317639-332254" indexRange="317639-317682"/>
  <SegmentURL mediaRange="332255-351547" indexRange="332255-332298"/>
  <SegmentURL mediaRange="351548-393022" indexRange="351548-351591"/>
  <SegmentURL mediaRange="393023-408236" indexRange="393023-393066"/>
  <SegmentURL mediaRange="408237-427369" indexRange="408237-408280"/>
  <SegmentURL mediaRange="427370-440107" indexRange="427370-427413"/>
  <SegmentURL mediaRange="440108-474282" indexRange="440108-440151"/>
  <SegmentURL mediaRange="474283-488057" indexRange="474283-474326"/>
  <SegmentURL mediaRange="488058-503247" indexRange="488058-488101"/>
  <SegmentURL mediaRange="503248-530342" indexRange="503248-503291"/>
  <SegmentURL mediaRange="530343-536661" indexRange="530343-530386"/>
  <SegmentURL mediaRange="536662-548842" indexRange="536662-536705"/>
  <SegmentURL mediaRange="548843-581169" indexRange="548843-548886"/>
  <SegmentURL mediaRange="581170-593467" indexRange="581170-581213"/>
  <SegmentURL mediaRange="593468-604261" indexRange="593468-593511"/>
  <SegmentURL mediaRange="604262-613484" indexRange="604262-604305"/>
  <SegmentURL mediaRange="613485-643872" indexRange="613485-613528"/>
  <SegmentURL mediaRange="643873-644861" indexRange="643873-643916"/>
  <SegmentURL mediaRange="644862-645417" indexRange="644862-644905"/>
</SegmentList>
</Representation>
```

Figura 12: *Bitrate* e segmentos do vídeo de resolução 480x360.

Como se pode observar na imagem acima, o vídeo de tamanho médio tem um *bitrate* de 316123 bit/s.

Vídeo de resolução 640x480:

```
<Representation id="3" mimeType="video/mp4" codecs="avc3.64001e" width="640" height="480" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="519755">
<BaseURL>videoB_640_480_1000k_dash.mp4</BaseURL>
<SegmentList timescale="15360" duration="7680">
  <SegmentURL mediaRange="927-21652" indexRange="927-970"/>
  <SegmentURL mediaRange="21653-45413" indexRange="21653-21696"/>
  <SegmentURL mediaRange="45414-74549" indexRange="45414-45457"/>
  <SegmentURL mediaRange="74550-124742" indexRange="74550-74593"/>
  <SegmentURL mediaRange="124743-150786" indexRange="124743-124786"/>
  <SegmentURL mediaRange="150787-180918" indexRange="150787-150830"/>
  <SegmentURL mediaRange="180919-235877" indexRange="180919-180962"/>
  <SegmentURL mediaRange="235878-266103" indexRange="235878-235921"/>
  <SegmentURL mediaRange="266104-309290" indexRange="266104-266147"/>
  <SegmentURL mediaRange="309291-347924" indexRange="309291-309334"/>
  <SegmentURL mediaRange="347925-413120" indexRange="347925-347968"/>
  <SegmentURL mediaRange="413121-450269" indexRange="413121-413164"/>
  <SegmentURL mediaRange="450270-489918" indexRange="450270-450313"/>
  <SegmentURL mediaRange="489919-556113" indexRange="489919-489962"/>
  <SegmentURL mediaRange="556114-581904" indexRange="556114-556157"/>
  <SegmentURL mediaRange="581905-617472" indexRange="581905-581948"/>
  <SegmentURL mediaRange="617473-682595" indexRange="617473-617516"/>
  <SegmentURL mediaRange="682596-708295" indexRange="682596-682639"/>
  <SegmentURL mediaRange="708296-735458" indexRange="708296-708339"/>
  <SegmentURL mediaRange="735459-756560" indexRange="735459-735502"/>
  <SegmentURL mediaRange="756561-807491" indexRange="756561-756604"/>
  <SegmentURL mediaRange="807492-830142" indexRange="807492-807535"/>
  <SegmentURL mediaRange="830143-853618" indexRange="830143-830186"/>
  <SegmentURL mediaRange="853619-892908" indexRange="853619-853662"/>
  <SegmentURL mediaRange="892909-902364" indexRange="892909-892952"/>
  <SegmentURL mediaRange="902365-919879" indexRange="902365-902408"/>
  <SegmentURL mediaRange="919880-967704" indexRange="919880-919923"/>
  <SegmentURL mediaRange="967705-984325" indexRange="967705-967748"/>
  <SegmentURL mediaRange="984326-1000868" indexRange="984326-984369"/>
  <SegmentURL mediaRange="1000869-1015282" indexRange="1000869-1000912"/>
  <SegmentURL mediaRange="1015283-1059261" indexRange="1015283-1015326"/>
  <SegmentURL mediaRange="1059262-1060510" indexRange="1059262-1059305"/>
  <SegmentURL mediaRange="1060511-1061166" indexRange="1060511-1060554"/>
</SegmentList>
</Representation>
</AdaptationSet>
</Period>
</MPD>
```

Figura 13: *Bitrate* e segmentos do vídeo de resolução 640x480.

Como se pode observar na imagem acima, o maior vídeo tem um *bitrate* de 519755 bit/s.

Para um cliente de *streaming* receber um vídeo através do *firefox*, terá de ter uma largura entre o vídeo pretendido e o vídeo seguinte, à exceção do último. Portanto, para o cliente observar um vídeo no *firefox* terá que conseguir ver o menor vídeo, ou seja, ter uma largura de banda entre 113040 bit/s e 316123 bit/s.

Para poder ver o vídeo de tamanho médio (480x360), terá que ter entre 316123 bit/s e 519755 bit/s.

Para ver o maior vídeo terá que ter uma largura de banda maior que 519755 bit/s, no caso do *core* uma vez que é uma simulação, pode-se deixar a largura de banda como ilimitada, de modo a transmitir sempre o maior vídeo.

A pilha protocolar utilizada neste contexto é a seguinte:

- Camada de aplicação (*Application Layer*) - HTTP (*Hypertext Transfer Protocol*)
- Camada de transporte (*Transport Layer*) - TCP (*Transmission Control Protocol*)
- Camada de rede (*Network Layer*) - IP (*Internet Protocol*)
- Camada de ligação (*Link Layer*) - (*Ethernet II*)

Pode-se verificar a camadas da pilha na seguinte imagem:


```

> Frame 18: 381 bytes on wire (3048 bits), 381 bytes captured (3048 bits) on interface veth5.0.8b, id 0
> Ethernet II, Src: 00:00:00_aa:00:08 (00:00:00:aa:00:08), Dst: 00:00:00_aa:00:00 (00:00:00:aa:00:00)
> Internet Protocol Version 4, Src: 10.0.2.20, Dst: 10.0.0.10
> Transmission Control Protocol, Src Port: 52350, Dst Port: 9999, Seq: 1, Ack: 1, Len: 315
> Hypertext Transfer Protocol

```

Figura 14: Pilha Protocolar usada no cenário da Etapa 2

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 Aladdin exiba o vídeo com mais resolução. Mostre evidências.

Como mencionado na questão anterior, de modo ao portátil da Bela ser forçado a exibir o vídeo de menor resolução (200x150), é necessário alterar a largura de banda para um valor 113040 bit/s e 316123 bit/s. Optou-se por escolher o 250000 bit/s como demonstrado na seguinte imagem da topologia *core*:

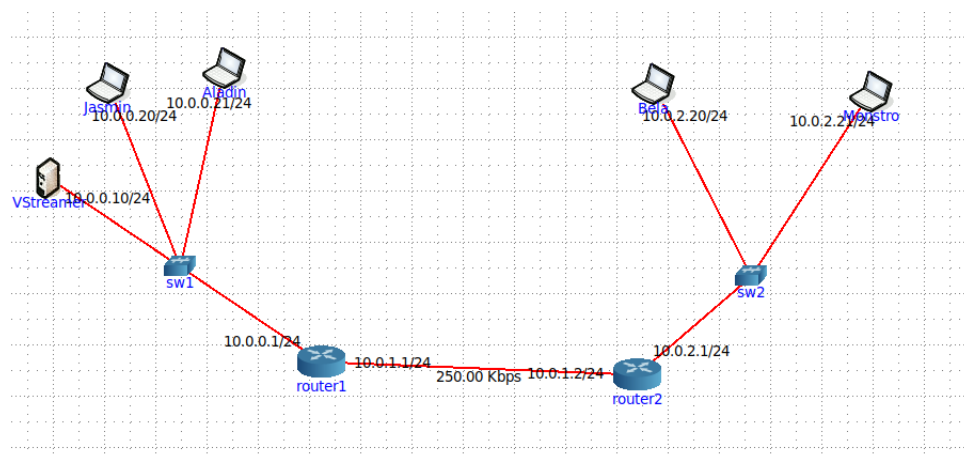


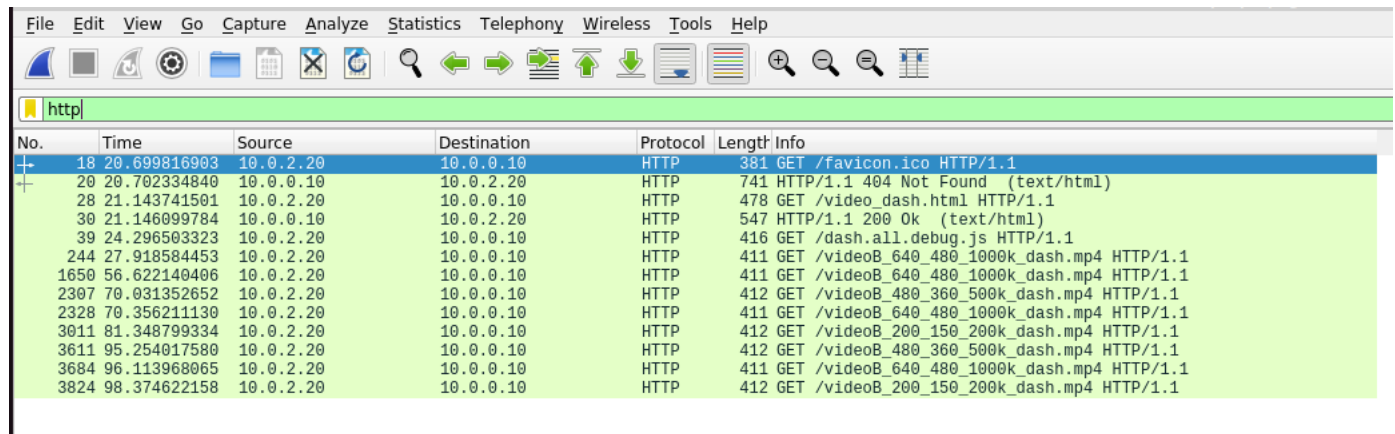
Figura 15: Topologia *core* ajustada à Questão 3

No caso da Bela, alterou-se o valor da taxa de transferência do *link* entre o router1 e o router2, para simular um ambiente com menor largura de banda.



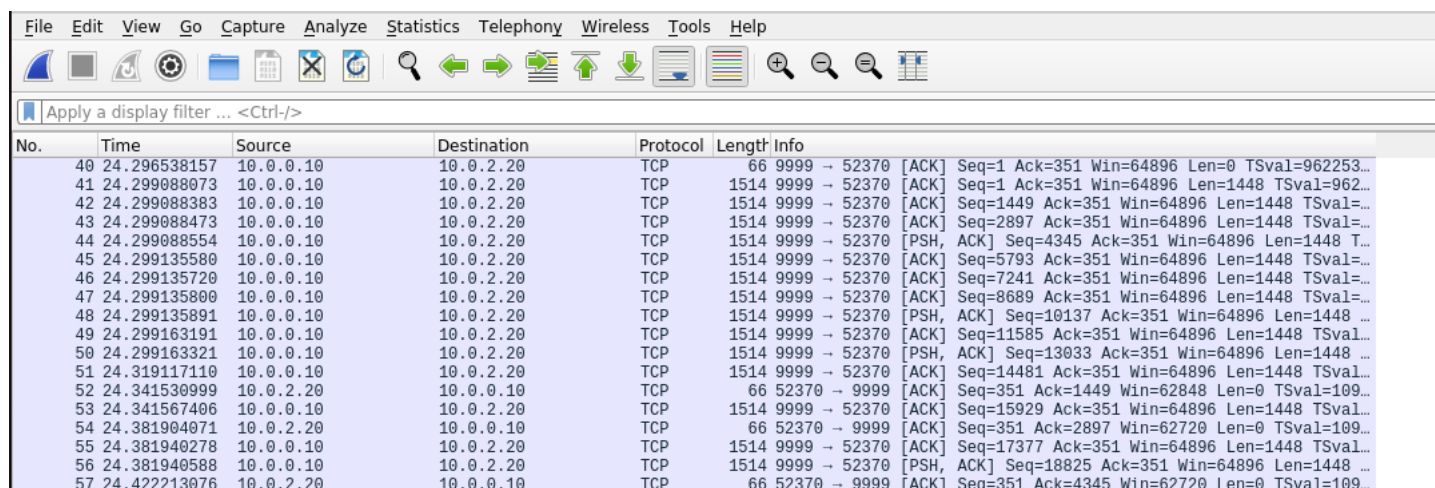
Figura 16: *Print* do *firefox* da Bela

Relativamente às evidências do *Wireshark*, infelizmente não foi possível captar um exemplo do bom funcionamento do DASH, já que grande parte das tentativas levavam a protocolos HTTP com informação *get* dos vídeo da ordem do maior para o menor iterativamente, ou seja, em ciclo, o que não é o esperado, como se vê na seguinte imagem:



No.	Time	Source	Destination	Protocol	Length	Info
18	20.699816903	10.0.2.20	10.0.0.10	HTTP	381	GET /favicon.ico HTTP/1.1
20	20.702334840	10.0.0.10	10.0.2.20	HTTP	741	HTTP/1.1 404 Not Found (text/html)
28	21.143741501	10.0.2.20	10.0.0.10	HTTP	478	GET /video_dash.html HTTP/1.1
30	21.146099784	10.0.0.10	10.0.2.20	HTTP	547	HTTP/1.1 200 Ok (text/html)
39	24.296503323	10.0.2.20	10.0.0.10	HTTP	416	GET /dash.all.debug.js HTTP/1.1
244	27.918584453	10.0.2.20	10.0.0.10	HTTP	411	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
1650	56.622140406	10.0.2.20	10.0.0.10	HTTP	411	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
2307	70.031352652	10.0.2.20	10.0.0.10	HTTP	412	GET /videoB_480_360_500k_dash.mp4 HTTP/1.1
2328	70.356211130	10.0.2.20	10.0.0.10	HTTP	411	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
3011	81.348799334	10.0.2.20	10.0.0.10	HTTP	412	GET /videoB_200_150_200k_dash.mp4 HTTP/1.1
3611	95.254017580	10.0.2.20	10.0.0.10	HTTP	412	GET /videoB_480_360_500k_dash.mp4 HTTP/1.1
3684	96.113968065	10.0.2.20	10.0.0.10	HTTP	411	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
3824	98.374622158	10.0.2.20	10.0.0.10	HTTP	412	GET /videoB_200_150_200k_dash.mp4 HTTP/1.1

Figura 17: *Print* do *Wireshark* da transmissão para a Bela



No.	Time	Source	Destination	Protocol	Length	Info
40	24.296538157	10.0.0.10	10.0.2.20	TCP	66	9999 → 52370 [ACK] Seq=1 Ack=351 Win=64896 Len=0 TSval=962253...
41	24.299088073	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=1 Ack=351 Win=64896 Len=1448 TSval=962...
42	24.299088383	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=1449 Ack=351 Win=64896 Len=1448 TSval=...
43	24.299088473	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=2897 Ack=351 Win=64896 Len=1448 TSval=...
44	24.299088554	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [PSH, ACK] Seq=4345 Ack=351 Win=64896 Len=1448 T...
45	24.299135580	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=5793 Ack=351 Win=64896 Len=1448 TSval=...
46	24.299135720	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=7241 Ack=351 Win=64896 Len=1448 TSval=...
47	24.299135800	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=8689 Ack=351 Win=64896 Len=1448 TSval=...
48	24.299135891	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [PSH, ACK] Seq=10137 Ack=351 Win=64896 Len=1448 ...
49	24.299163191	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=11585 Ack=351 Win=64896 Len=1448 TSval=...
50	24.299163321	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [PSH, ACK] Seq=13033 Ack=351 Win=64896 Len=1448 ...
51	24.319117110	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=14481 Ack=351 Win=64896 Len=1448 TSval=...
52	24.341530999	10.0.2.20	10.0.0.10	TCP	66	52370 → 9999 [ACK] Seq=351 Ack=1449 Win=62848 Len=0 TSval=109...
53	24.341567406	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=15929 Ack=351 Win=64896 Len=1448 TSval=...
54	24.381904071	10.0.2.20	10.0.0.10	TCP	66	52370 → 9999 [ACK] Seq=351 Ack=2897 Win=62720 Len=0 TSval=109...
55	24.381940278	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [ACK] Seq=17377 Ack=351 Win=64896 Len=1448 TSval=...
56	24.381940588	10.0.0.10	10.0.2.20	TCP	1514	9999 → 52370 [PSH, ACK] Seq=18825 Ack=351 Win=64896 Len=1448 ...
57	24.422213076	10.0.2.20	10.0.0.10	TCP	66	52370 → 9999 [ACK] Seq=351 Ack=4345 Win=62720 Len=0 TSval=109...

Figura 18: *Print* do *Wireshark* da transmissão para a Bela 2

No caso Aladin, como mencionado também na questão anterior, não é necessário alterar o *link*, uma vez que o *default* é ter largura de banda ilimitada, e por isso, o Aladin terá o maior vídeo exibido pelo *firefox*.



Figura 19: *Print do firefox do Aladin*

No.	Time	Source	Destination	Protocol	Length	Info
13	7.233997111	10.0.0.21	10.0.0.10	HTTP	478	GET /video_dash.html HTTP/1.1
15	7.234173884	10.0.0.10	10.0.0.21	HTTP	547	HTTP/1.1 200 Ok (text/html)
21	7.648978592	10.0.0.21	10.0.0.10	HTTP	381	GET /favicon.ico HTTP/1.1
23	7.649121451	10.0.0.10	10.0.0.21	HTTP	741	HTTP/1.1 404 Not Found (text/html)
31	8.387329490	10.0.0.21	10.0.0.10	HTTP	416	GET /dash.all.debug.js HTTP/1.1
2250	8.560302607	10.0.0.10	10.0.0.21	HTTP	254	HTTP/1.1 200 Ok (application/x-javascript)
2258	8.939449880	10.0.0.21	10.0.0.10	HTTP	411	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
3029	8.972228454	10.0.0.10	10.0.0.21	MP4	1503	
3036	9.283125386	10.0.0.21	10.0.0.10	HTTP	413	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
3804	9.325138375	10.0.0.10	10.0.0.21	MP4	1503	
3813	9.612208956	10.0.0.21	10.0.0.10	HTTP	413	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
4593	9.634118126	10.0.0.10	10.0.0.21	MP4	1503	
4600	9.836699625	10.0.0.21	10.0.0.10	HTTP	414	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
5373	9.877594989	10.0.0.10	10.0.0.21	MP4	1503	
5381	10.070283324	10.0.0.21	10.0.0.10	HTTP	415	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
6153	10.097899213	10.0.0.10	10.0.0.21	MP4	1503	
6160	10.215250526	10.0.0.21	10.0.0.10	HTTP	415	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
7073	10.232308619	10.0.0.10	10.0.0.21	MP4	1503	
7080	10.327599076	10.0.0.21	10.0.0.10	HTTP	415	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
7859	10.433851102	10.0.0.10	10.0.0.21	MP4	1503	
7866	10.771014673	10.0.0.21	10.0.0.10	HTTP	415	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1
8638	10.806576770	10.0.0.10	10.0.0.21	MP4	1503	
8645	10.986762003	10.0.0.21	10.0.0.10	HTTP	415	GET /videoB_640_480_1000k_dash.mp4 HTTP/1.1

Figura 20: *Print do Wireshark da transmissão para a Aladin*

Como a largura de banda padrão é ilimitada, o DASH busca sempre o fragmento do maior vídeo.

No.	Time	Source	Destination	Protocol	Length	Info
34	8.387523606	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=1449 Ack=351 Win=64896 Len=1448 TSval=...
35	8.387523706	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=2897 Ack=351 Win=64896 Len=1448 TSval=...
36	8.387523786	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=4345 Ack=351 Win=64896 Len=1448 TSval=...
37	8.387523876	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [PSH, ACK] Seq=5793 Ack=351 Win=64896 Len=1448 T...
38	8.387590822	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=7241 Ack=351 Win=64896 Len=1448 TSval=...
39	8.387591013	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=8689 Ack=351 Win=64896 Len=1448 TSval=...
40	8.387591103	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=10137 Ack=351 Win=64896 Len=1448 TSval=...
41	8.387591193	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=11585 Ack=351 Win=64896 Len=1448 TSval=...
42	8.387591283	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [PSH, ACK] Seq=13033 Ack=351 Win=64896 Len=1448 ...
43	8.387675933	10.0.0.21	10.0.0.10	TCP	66	55014 → 9999 [ACK] Seq=351 Ack=2897 Win=61440 Len=0 TSval=501...
44	8.387685711	10.0.0.21	10.0.0.10	TCP	1514	9999 → 55014 [ACK] Seq=14481 Ack=351 Win=64896 Len=1448 TSval=...
45	8.387685831	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=15929 Ack=351 Win=64896 Len=1448 TSval=...
46	8.387685922	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=17377 Ack=351 Win=64896 Len=1448 TSval=...
47	8.387686012	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [PSH, ACK] Seq=18825 Ack=351 Win=64896 Len=1448 ...
48	8.387699918	10.0.0.21	10.0.0.10	TCP	66	55014 → 9999 [ACK] Seq=351 Ack=5793 Win=58624 Len=0 TSval=501...
49	8.387704226	10.0.0.21	10.0.0.10	TCP	66	55014 → 9999 [ACK] Seq=351 Ack=8689 Win=55808 Len=0 TSval=501...
50	8.387721358	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=20273 Ack=351 Win=64896 Len=1448 TSval=...
51	8.387721499	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=21721 Ack=351 Win=64896 Len=1448 TSval=...
52	8.387721619	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=23169 Ack=351 Win=64896 Len=1448 TSval=...
53	8.387721729	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=24617 Ack=351 Win=64896 Len=1448 TSval=...
54	8.387721829	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=26065 Ack=351 Win=64896 Len=1448 TSval=...
55	8.387721929	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=27513 Ack=351 Win=64896 Len=1448 TSval=...
56	8.387722030	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [ACK] Seq=28961 Ack=351 Win=64896 Len=1448 TSval=...
57	8.387722130	10.0.0.10	10.0.0.21	TCP	1514	9999 → 55014 [PSH, ACK] Seq=30409 Ack=351 Win=64896 Len=1448 ...

Figura 21: *Print do Wireshark da transmissão para a Aladin 2*

Questão 4: Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado.

O DASH (*Dynamic, Adaptative Streaming over HTTP*) é uma técnica que permite adaptar o conteúdo transmitido à taxa de transferência às características da rede, durante o *streaming* de multimédia via HTTP.

Esta funcionalidade baseia-se na segmentação dos dados a transferir em diferentes gamas, com diferentes tamanhos, dependendo da taxa disponível. Quando o cliente repara que a rede não consegue suportar mais a multimédia que requer maior taxa, pede ao servidor um de taxa menor, de dentro dos disponíveis.

Neste caso específico, o ficheiro *video_manifest.mpd* (media presentation description), criado com referência aos três vídeos de diferentes resoluções desta etapa, detalha os diferentes ficheiros multimédia disponíveis para o *streaming*, as gamas dos seus segmentos e a largura de banda mínima para a sua escolha (e efetiva transmissão). As imagens 11, 12 e 13 demonstram as diferentes partes deste ficheiro, com os detalhes referidos presentes. O DASH tem acesso a este ficheiro, como se pode ver no *html* da página acedida, informando os pedidos do cliente ao servidor.

```

<!DOCTYPE html>
<html>
  <head>
    <title> Streaming ESR 2023 </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 22: Código-fonte da página *video_dash.html*.

Nas capturas junto do servidor, verifica-se um comportamento diferente do expectado, como mencionado na Questão 3. Teoricamente, segundo informações discutidas com o docente das aulas práticas, o processo de pedido do ficheiro é feito do seguinte modo: Inicialmente, é feito um *get* ganancioso do maior ficheiro e, posteriormente, depois de adquirir informação sobre a rede, deve ser feito, em ciclo, *get* do ficheiro mais apropriado, tal como do seguinte ficheiro mais pesado, testando a sua possibilidade de transmissão.

Os resultados obtidos estão representados na figura 17, onde se repara, ao invés do descrito anteriormente, na indecisão do protocolo em determinar qual dos ficheiros utilizar, quando o limite da rede apenas permite o ficheiro mais pequeno. De qualquer forma, é notável a tentativa de primeiro transmitir o vídeo mais exigente e também as sucessivas reduções da qualidade do ficheiro pedido em resposta às taxas da rede, ainda que não seja claro qual o critério para tal, que estabilizaram naquele passível de ser transmitido, como seria de esperar.

Etapa 3

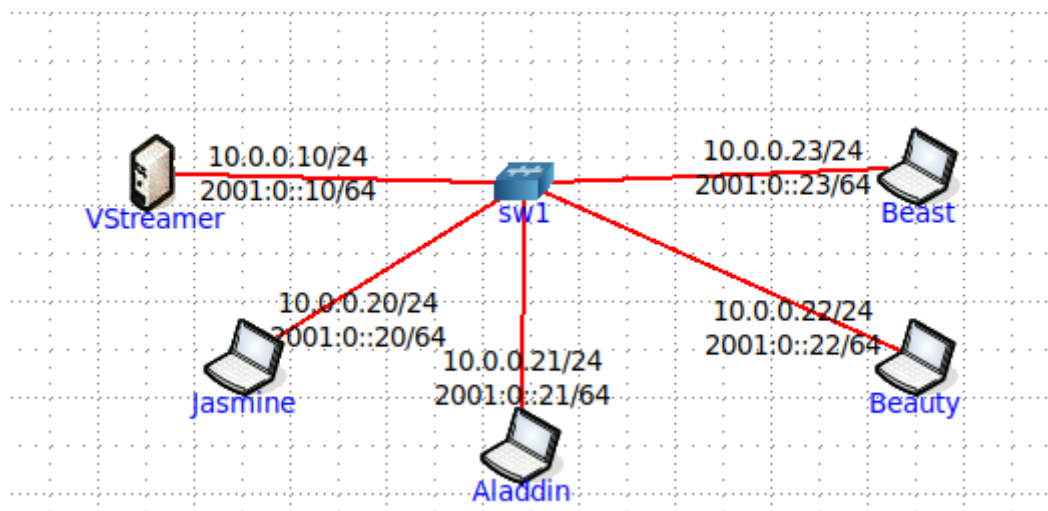


Figura 23: Topologia do cenário *Multicast*

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.

Como descrito no enunciado, o cenário *unicast* envia o conjunto de pacotes necessários para cada um dos clientes, enquanto o *multicast* envia apenas um conjunto e a rede replica esse conjunto para todos os intervenientes do grupo.

Note-se a taxa de aproximadamente 230 kbit/s utilizada para transferir para um utilizador:

Wireshark - Protocol Hierarchy Statistics - pl21_tp1_ex3_udp_singlecast.pcapng									
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	
Frame	100.0	730	100.0	592350	230 k	0	0	0	
Ethernet	100.0	730	1.7	10220	3981	0	0	0	
Internet Protocol Version 6	0.5	4	0.0	160	62	0	0	0	
User Datagram Protocol	0.1	1	0.0	8	3	0	0	0	
Multicast Domain Name System	0.1	1	0.0	45	17	1	45	17	
Open Shortest Path First	0.3	2	0.0	72	28	2	72	28	
Internet Control Message Protocol v6	0.1	1	0.0	16	6	1	16	6	
Internet Protocol Version 4	99.2	724	2.4	14480	5640	0	0	0	
User Datagram Protocol	97.8	714	1.0	5712	2225	0	0	0	
Data	97.1	709	94.6	560457	218 k	709	560457	218 k	
ADwin configuration protocol	0.7	5	0.1	684	266	5	684	266	
Open Shortest Path First	1.4	10	0.1	440	171	10	440	171	
Address Resolution Protocol	0.3	2	0.0	56	21	2	56	21	

Figura 24: Estatísticas do cenário *unicast* com 1 utilizador.

Wireshark - Conversations - pl21_tp1_ex3_udp_singlecast.pcapng											
Ethernet · 5		IPv4 · 2	IPv6 · 3	TCP	UDP · 3						
Address A	Address 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
10.0.0.1	224.0.0.5	10	780	10	780	0	0	0.984847	18.0127	346	
10.0.0.10	10.0.2.21	714	591 k	714	591 k	0	0	0.000000	20.5355	230 k	

Figura 25: Conversações do cenário *unicast* com 1 utilizador.

Para testar a escalabilidade, decidiu-se realizar as transmissões *unicast* com mais utilizadores.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	1407	100.0	1249131	485 k	0	0	0
Ethernet	100.0	1407	1.6	19698	7650	0	0	0
Internet Protocol Version 6	0.1	2	0.0	80	31	0	0	0
Open Shortest Path First	0.1	2	0.0	72	27	2	72	27
Internet Protocol Version 4	99.7	1403	2.2	28060	10 k	0	0	0
User Datagram Protocol	99.0	1393	0.9	11144	4327	0	0	0
Data	98.4	1384	95.2	1188829	461 k	1384	1188829	461 k
ADwin configuration protocol	0.6	9	0.1	752	292	9	752	292
Open Shortest Path First	0.7	10	0.0	440	170	10	440	170
Address Resolution Protocol	0.1	2	0.0	56	21	2	56	21

Figura 26: Estatísticas do cenário *unicast* com 2 utilizadores.

Wireshark - Conversations - pl21_tp1_ex3_udp_singlecast_2users.pcapng											
Ethernet · 4		IPv4 · 3	IPv6 · 1	TCP	UDP · 4						
Address A	Address 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
10.0.0.1	224.0.0.5	10	780	10	780	0	0	1.541163	18.0071		346
10.0.0.10	10.0.0.20	668	634 k	668	634 k	0	0	0.000000	20.5967		246 k
10.0.0.10	10.0.2.21	725	613 k	725	613 k	0	0	0.000441	20.5987		238 k

Figura 27: Conversações do cenário *unicast* com 2 utilizadores.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	2178	100.0	1779405	672 k	0	0	0
Ethernet	100.0	2178	1.7	30492	11 k	0	0	0
Internet Protocol Version 6	0.1	2	0.0	80	30	0	0	0
Open Shortest Path First	0.1	2	0.0	72	27	2	72	27
Internet Protocol Version 4	99.9	2176	2.4	43520	16 k	0	0	0
User Datagram Protocol	99.4	2165	1.0	17320	6549	0	0	0
Data	99.1	2159	94.8	1686977	637 k	2159	1686977	637 k
ADwin configuration protocol	0.3	6	0.0	460	173	6	460	173
Open Shortest Path First	0.5	11	0.0	484	183	11	484	183

Figura 28: Estatísticas do cenário *unicast* com 3 utilizadores.

Wireshark - Conversations - veth5.0.6b											
Ethernet · 5		IPv4 · 4		IPv6 · 1		TCP		UDP · 6			
Address A	Address 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
10.0.0.1	224.0.0.5	11	858	11	858	0	0	0.087928	20.0068		343
10.0.0.10	10.0.0.21	738	598 k	738	598 k	0	0	0.000000	21.1450		226 k
10.0.0.10	10.0.2.21	755	654 k	755	654 k	0	0	0.000375	21.1548		247 k
10.0.0.10	10.0.0.20	672	525 k	672	525 k	0	0	0.000693	21.1449		198 k

Figura 29: Conversações do cenário *unicast* com 3 utilizadores.

No total, o servidor necessitou de uma largura de banda de 461 kbit/s para 2 clientes, aproximadamente o dobro da taxa necessária para um só cliente e 637 kbit/s para 3 clientes, aproximadamente o triplo. Analisadas as conversações, para cada um dos clientes foram enviados *bits* com uma taxa semelhante à que foi enviada no cenário com um único utilizador. Pode-se assumir então um aumento linear da necessidade de largura de banda com o crescimento do número de utilizadores. Isto representa um problema de escalabilidade.

Também se repare na quantidade de *bytes* enviados para cada um dos utilizadores. O servidor está a enviar para cada um deles o vídeo na sua totalidade, o que implica repetição dos dados na rede, nomeadamente entre o servidor e o *switch* local. Pressupõe-se que esta seja a razão pela necessidade de uma maior largura de rede, mencionada anteriormente.

Posteriormente testou-se o cenário *multicast*, com atenção ao tráfego de rede.

Neste caso a taxa de transferência e a quantidade total de *bytes* enviadas é aproximadamente igual ao caso de um único utilizador em *unicast*.

Wireshark - Protocol Hierarchy Statistics - pl21_tp1_ex3_udp_multicast.pcapng								
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	822	100.0	720050	214 k	0	0	0
▼ Ethernet	100.0	822	1.6	11508	3429	0	0	0
▼ Internet Protocol Version 4	100.0	822	2.3	16440	4899	0	0	0
▼ User Datagram Protocol	100.0	822	0.9	6576	1959	0	0	0
▼ Session Announcement Protocol	0.6	5	0.2	1620	482	0	0	0
Session Description Protocol	0.6	5	0.2	1500	447	5	1500	447
▼ Real-Time Transport Protocol	88.3	726	89.0	641002	191 k	0	0	0
MP4V-ES	88.3	726	87.8	632290	188 k	726	632290	188 k
Real-time Transport Control Protocol	0.6	5	0.0	140	41	5	140	41
Data	10.3	85	5.9	42660	12 k	85	42660	12 k
ADwin configuration protocol	0.1	1	0.0	104	30	1	104	30

Figura 30: Estatísticas do cenário *multicast*.

Wireshark - Conversations - pl21_tp1_ex3_udp_multicast.pcapng											
Ethernet · 2	IPv4 · 2	IPv6	TCP	UDP · 3							
Address A	Address 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	224.0.0.200	817	718 k	817	718 k	0	0	0.000000	26.8443	214 k	
10.0.0.10	224.2.127.254	5	1830	5	1830	0	0	3.290853	20.1045	728	

Figura 31: Conversações do cenário *multicast*.

Também se testou o cenário *multicast* apenas com um cliente conectado, de forma a comparar o caso mais simples entre este e o *unicast*.

Wireshark - Protocol Hierarchy Statistics - veth5.0.6b								
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	733	100.0	676686	237 k	0	0	0
▼ Ethernet	100.0	733	1.5	10262	3608	0	0	0
▼ Internet Protocol Version 6	0.4	3	0.0	120	42	0	0	0
Open Shortest Path First	0.4	3	0.0	108	37	3	108	37
▼ Internet Protocol Version 4	99.6	730	2.2	14600	5133	0	0	0
▼ User Datagram Protocol	98.1	719	0.9	5752	2022	0	0	0
▼ Session Announcement Protocol	0.7	5	0.2	1620	569	0	0	0
Session Description Protocol	0.7	5	0.2	1500	527	5	1500	527
▼ Real-Time Transport Protocol	90.6	664	90.3	611013	214 k	0	0	0
MP4V-ES	90.6	664	89.1	603045	212 k	664	603045	212 k
Real-time Transport Control Protocol	0.7	5	0.0	140	49	5	140	49
Data	6.0	44	4.8	32491	11 k	44	32491	11 k
ADwin configuration protocol	0.1	1	0.0	96	33	1	96	33
Open Shortest Path First	1.5	11	0.1	484	170	11	484	170

Figura 32: Estatísticas do cenário *multicast* com 1 cliente.

Wireshark - Conversations - veth5.0.6b											
Ethernet · 4	IPv4 · 3	IPv6 · 1	TCP	UDP · 3							
Address A	Address 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
10.0.0.1	224.0.0.5	11	858	11	858	0	0	1.476810	20.0098	343	
10.0.0.10	224.0.0.200	714	673 k	714	673 k	0	0	0.000000	22.7511	236 k	
10.0.0.10	224.2.127.254	5	1830	5	1830	0	0	1.656587	20.1031	728	

Figura 33: Conversações do cenário *multicast* com 1 cliente.

Neste cenário, foram enviados 714 *packets* para o grupo de transmissão, o mesmo número que os enviados no caso de 1 cliente em *unicast*, mas com um número mais elevado de *bytes* transmitidos, o que indica uma menor eficiência neste caso pelo seu maior peso e complexidade de protocolo.

Retiram-se assim as seguintes conclusões:

- A taxa de transferência necessária não aumenta em resposta ao aumento do número de utilizadores, o que demonstra uma maior capacidade de escalabilidade.
- O facto de se enviar apenas um pacote reduz o congestionamento de tráfego, visto que apenas um pacote é propagado pela rede.

- A maior complexidade dos pacotes integrantes do *multicast* representa um maior peso de rede que não compensa o seu uso no caso em que o servidor apenas envia dados a um cliente. *Unicast*, nesse cenário, é mais simples e eficiente.

Conclusão

Em suma,

Na Etapa 1, compreendemos o funcionamento do *streaming* sobre HTTP, e reparamos no seu problema de escalabilidade com um crescimento do número de utilizadores.

Na Etapa 2, embora tivéssemos dificuldades em presenciar a funcionalidade expectável do DASH, no que diz respeito à escolha iterativa dos ficheiros a partir da taxa, conseguimos entender a adaptabilidade do conteúdo transmitido relativamente às taxas, através do ficheiro *.mpd* gerado.

Na Etapa 3, aletradamo-nos tanto nas especificidades de, como nas diferenças entre, *streaming* em *unicast* e em *multicast*, do ponto de vista da largura de rede necessária e da sobrecarga do seu tráfego na rede, seja de um dos cenários como do outro.