

Escola de Engenharia
Universidade do Minho

Streaming de áudio e vídeo a pedido e em tempo real

Engenharia de Serviços em Rede
Mestrado em Engenharia Informática

Trabalho Prático 1

Diogo Rebelo (PG50327)
João Caldas (PG50494)
Teresa Fortes (PG50770)

20 de junho de 2023

Conteúdo

Motivação	4
Introdução	4
Etapa 1 - Streaming HTTP simples sem adaptação dinâmica de débito	6
Questão 1	6
Etapa 2 - Streaming adaptativo sobre HTTP (MPEG-DASH)	17
Questão 2	17
Questão 3	19
Questão 4	21
Etapa 3 - Streaming RTP/RTCP unicast sobre UDP e multicast com anúncios SAP	24
Questão 5	24
Conclusão	26

Lista de Figuras

1	Configuração da topologia requisitada	6
2	Taxa de transmissão esperado (em bps).	7
3	Taxa de transmissão real (em bps) - Jasmine.	7
4	Ilustração do fluxo de bits transmitidos por segundo.	8
5	Taxa de transmissão real (em bps) - Jasmine & Bela.	8
6	Taxa de transmissão real (em bps) - Jasmine, Bela & Monstro.	8
7	Encapsulamento protocolar com 1 cliente.	9
8	Encapsulamento protocolar com 2 clientes.	10
9	Encapsulamento protocolar com 3 clientes.	10
10	Encapsulamento protocolar nos pacotes (componentes)[5].	11
11	Fluxo gerado pelo cliente 1 - VStreamer.	12
12	Fluxo gerado pelos 2 clientes - Jasmine.	13
13	Tráfego entre os dois clientes - Jasmine e Bela.	14
14	Fluxo gerado pelos 4 clientes - Jasmine, Bela (x2) e Monstro.	16
15	Largura de banda do vídeo de menor resolução (160*100)	17
16	Largura de banda do vídeo de resolução intermédia (320*200)	17
17	Largura de banda do vídeo de maior resolução (640*400)	17
18	Pilha protocolar usada na transmissão do vídeo para o portátil Bela	18
19	Pilha protocolar usada na transmissão do vídeo para o portátil Alladin	18
20	Topologia com alteração da restrição do link para o portátil Bela	19
21	Captura da transmissão do vídeo para o portátil Bela	19
22	Topologia com alteração da restrição do link para o portátil Alladin	20
23	Captura da transmissão do vídeo para o portátil Alladin	20
24	Arquitetura DASH e fluxo de acordo com o exemplo anterior (os valores das taxas são meramente exemplificativos).	21
25	Funcionamento DASH e fluxo de acordo com o exemplo anterior (os valores das taxas são meramente exemplificativos).	22
26	Dados transmitidos pelo serviço <i>unicast</i>	24
27	Dados transmitidos pelo serviço <i>multicast</i>	24

Motivação

É curioso o facto de o *streaming* de conteúdo pela Internet se ter iniciado pelos anos 90, com o principal desafio de lidar com uma grande quantidade de dados e combater o *best-effort* da Internet, que não foi inicialmente projetada para suportar vídeos de alta qualidade.

Primeiramente, aparece o Protocolo de Transporte em Tempo Real (RTP) que prometia oferecer suporte ao *streaming* em tempo real, através do UDP [7], no entanto, verificavam-se problemas com NATs e firewalls e o servidor seria o gestor das sessões de *streaming* de cada cliente separadamente. Então o RTP foi sendo cada vez menos usado.

Rapidamente, foi-se optando por tecnologias que usavam TCP, surgindo então o *streaming* HTTP, que logo se tornou o mais utilizado pelas grandes empresas, levando a um aumento da preocupação com os protocolos e padrões utilizados, pois estes teriam um grande impacto na experiência do usuário (QoE).[8]

Introdução

No âmbito da unidade curricular de Engenharia de Serviços em Rede, do 1º ano do Mestrado em Engenharia Informática da Universidade do Minho, foi-nos proposta a elaboração de um projeto que abordasse o tema do *Streaming* de áudio e vídeo.

O presente relatório segue um formato e uma ordem de ideias específicos, de acordo com a metodologia que foi solicitada. Neste contexto, surge detalhado o procedimento e as respostas às respetivas questões, bem como aspetos que o grupo considerou relevantes para a completude do trabalho.

Para o desenvolvimento deste trabalho prático, socorremo-nos da máquina virtual Xubuncore, do Wireshark e de bibliografia relevante.

Naturalmente, existe um conjunto de objetivos primordiais, que se pretendem ver concretizados, nomeadamente:

1. Compreender o processo de *Streaming* de vídeo sobre HTTP com apenas um cliente ou com vários clientes ativos, percebendo o impacto do aumento do número de clientes na taxa de bits transmitidos por segundo (Escalabilidade);
2. Rever conceitos de encapsulamento protocolar, como forma de compreensão da estrutura dos pacotes transmitidos entre o servidor e os seus clientes;
3. Consolidar conteúdo relevante sobre o *Streaming* adaptativo sobre HTTP, através da compreensão da arquitetura e funcionamento do MPEG-DASH;
4. Ser capaz de distinguir e comparar os diferentes métodos de *streaming* abordados, tendo em conta a arquitetura cliente-servidor utilizada;
5. Distinguir cenários *unicast* vs. *multicast*, argumentando de acordo com as suas vantagens e desvantagens e efetuando a relação com os conceitos de escalabilidade e tráfego de rede.
6. Desenvolver competências ao nível do serviço aplicacional em questão, isto é, aprofundar conhecimentos sobre ferramentas de manipulação de vídeo,

utilizando-as na emulação do *Core* para a obtenção de resultados eficazes e concisos;

Etapa 1 - Streaming HTTP simples sem adaptação dinâmica de débito

Segue-se o enunciado de cada questão, seguido do procedimento utilizado para responder à mesma. Ao longo desta secção, procuramos responder de modo completo a cada uma das perguntas, evidenciando aspectos que consideramos relevantes.

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).

Começamos por gerar os dois vídeos solicitados, através do `ffmpeg` com as características requisitadas, e passamos à construção da topologia do enunciado, cuja configuração se segue:

- 4 portáteis;
- 1 servidor;
- 2 switches;
- 2 routers;

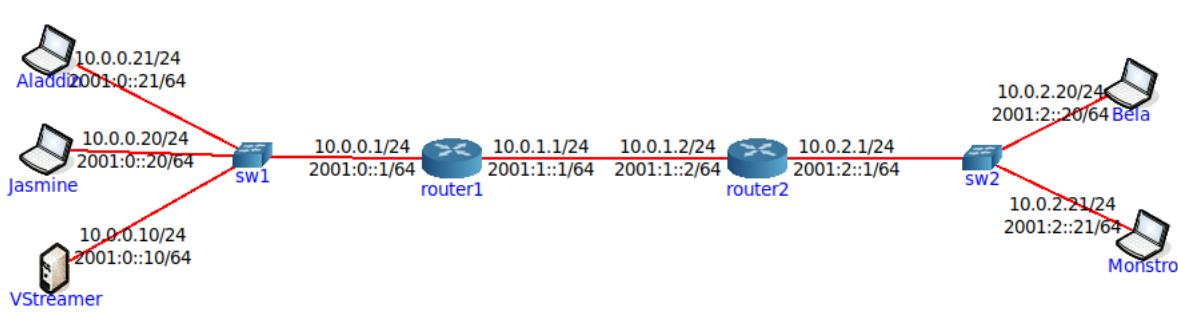


Figura 1: Configuração da topologia requisitada

Ao longo desta etapa, vamos efetuando *Streaming* apenas por HTTP simples e sem adaptação de débito, sendo o objetivo verificar as características do tráfego gerado na rede, quer com 1, com 2 ou 3 clientes em simultâneo.

- Taxa (em bits por segundo) de transmissão necessária:

Através do comando `ffmpeg -i videoA.mp4`, procuramos pela antepenúltima linha, no terminal especificado na figura 2 e observamos, logo no inicio da linha, valor de 11 kbps. Contudo, não vindo esta informação de um analisador de tráfego, consideramos este valor como um valor teórico ideal e, por isso, uma taxa de transmissão ideal.

```

Terminal - core@xubuncore: ~
File Edit View Terminal Tabs Help
-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 : Lavf58.29.100
  Duration: 00:00:13.95, start: 0.000000, bitrate: 13 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 160x100,
  11 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
  Metadata:
    handler_name : VideoHandler
At least one output file must be specified
core@xubuncore:~$ 

```

Figura 2: Taxa de transmissão esperado (em bps).

Passemos, então à análise da taxa de transmissão real, para cada um dos casos (para 1, 2 e 3 clientes).

(1) Com apenas um cliente (VLC):

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
Frame	100.0	144	100.0	93149	91 k	0
Ethernet	100.0	144	2.2	2016	1984	0
Internet Protocol Version 6	0.7	1	0.0	40	39	0
Open Shortest Path First	0.7	1	0.0	36	35	1
Internet Protocol Version 4	97.9	141	3.0	2820	2775	0
Transmission Control Protocol	94.4	136	94.4	87961	86 k	125
Hypertext Transfer Protocol	7.6	11	17.7	16462	16 k	11
Open Shortest Path First	3.5	5	0.2	220	216	5
Address Resolution Protocol	1.4	2	0.1	56	55	2

Figura 3: Taxa de transmissão real (em bps) - Jasmine.

Como concluímos na figura anterior, na linha destacada a azul e na coluna Bits/s, a taxa que realmente foi medida foi de 16 kbps, uma taxa superior à esperada. Verifica-se uma percentagem de erro igual a $(16 - 11)/11) * 100 = 45\% (45\%)$. Esta percentagem pode ser explicada pela perda de pacotes aquando da transmissão, devido à formação errada de pacotes.

Ainda no gráfico seguinte, que descreve o fluxo dos bits transmitidos por segundo, observamos que o tráfego relativo ao vídeo foi transmitido algures entre os 4 e 8 segundos, porque se verifica um pico de transmissão sensivelmente perto dos 16 000 bits/s (esta relação também se verifica para os restantes casos). Note-se que o tráfego surge filtrado pelo protocolo HTTP.

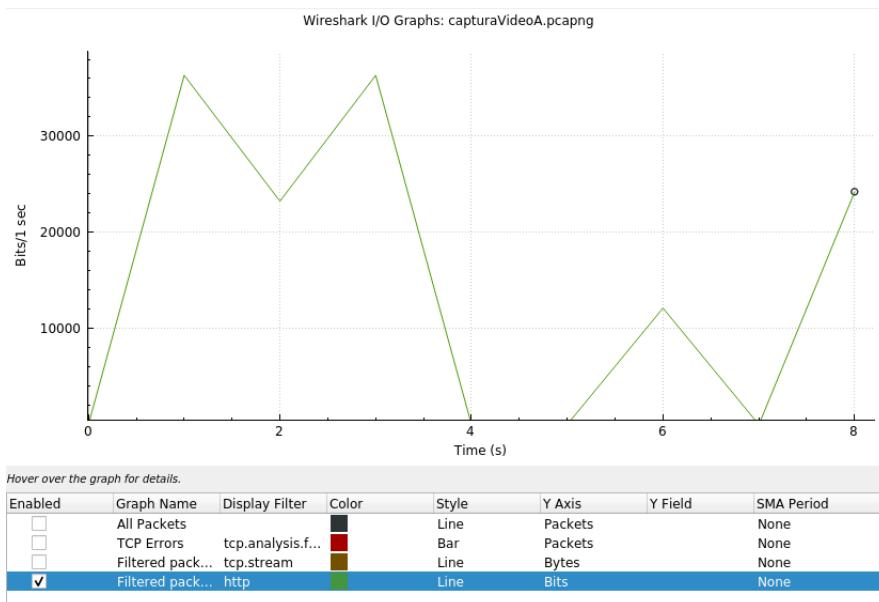


Figura 4: Ilustração do fluxo de bits transmitidos por segundo.

(2) Com dois clientes (VLC e Firefox):

Do mesmo modo, podemos analisar a taxa de transmissão através do *Wireshark*, e desta vez, verificamos uma taxa de transmissão superior, ou seja, de 20 kbps:

Protocol	Percent Packets	packets	Percent Bytes	Bytes	Bits/s	End Pac
Frame	100.0	513	100.0	333366	174 k	0
Ethernet	100.0	513	2.2	7182	3750	0
Internet Protocol Version 6	0.8	4	0.0	160	83	0
Open Shortest Path First	0.4	2	0.0	72	37	2
Internet Control Message Prot...	0.4	2	0.0	32	16	2
Internet Protocol Version 4	98.4	505	3.0	10100	5274	0
Transmission Control Protocol	96.9	497	94.6	315356	164 k	469
Hypertext Transfer Protocol	5.5	28	11.9	39637	20 k	28
Open Shortest Path First	1.6	8	0.1	352	183	8
Address Resolution Protocol	0.8	4	0.0	112	58	4

Figura 5: Taxa de transmissão real (em bps) - Jasmine & Bela.

(3) Com três clientes (VLC, Firefox e ffplay):

A taxa verificada, desta vez, foi de 32 kbps, como se verifica na figura seguinte. Mais uma vez, uma taxa superior às anteriores.

Protocol	Percent Packets	packets	Percent Bytes	Bytes	Bits/s	End Packets
Frame	100.0	1080	100.0	722762	261 k	0
Ethernet	100.0	1080	2.1	15120	5465	0
Internet Protocol Version 6	0.2	2	0.0	80	28	0
Open Shortest Path First	0.2	2	0.0	72	26	2
Internet Protocol Version 4	99.8	1078	3.0	21560	7793	0
Transmission Control Protocol	98.8	1067	94.8	685446	247 k	1004
Hypertext Transfer Protocol	5.8	63	12.6	90806	32 k	63
Open Shortest Path First	1.0	11	0.1	484	174	11

Figura 6: Taxa de transmissão real (em bps) - Jasmine, Bela & Monstro.

Deste modo, conclui-se que parece existir uma relação diretamente proporcional entre o número de streaming Clients e a taxa real de transmissão de pacotes,

em bits por segundo. É natural que assim seja, já que é necessário transmitir mais bits para garantir a chegada dos dados a cada cliente.

Contudo, verificamos que, por exemplo, no caso do cliente no Firefox, alguns frames do respetivo vídeo eram perdidos e o vídeo não era visualizado na sua totalidade. Ora, isto pode comprovar a perda de pacotes, os quais não parecem chegar a cada cliente ao da mesma forma: o servidor aparentava atender a pedidos individualmente, sendo que, enquanto um vídeo se visualizava sem problemas, os outros congelavam, mesmo que por poucos segundos.

- **Encapsulamento do pacotes transmitidos:**

Observamos que o processo de encapsulamento ocorre ao longo dos vários níveis da pilha protocolar (tabela 1) e que em cada nível há um protocolo específico, evidenciado na aba de detalhes do tráfego, no Wireshark (figuras 7, 8 e 9).

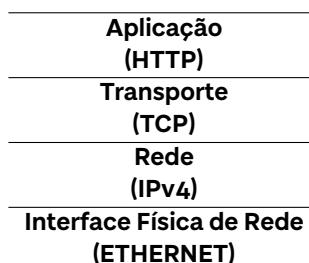


Tabela 1: Pilha protocolar, modelo TCP/IP.

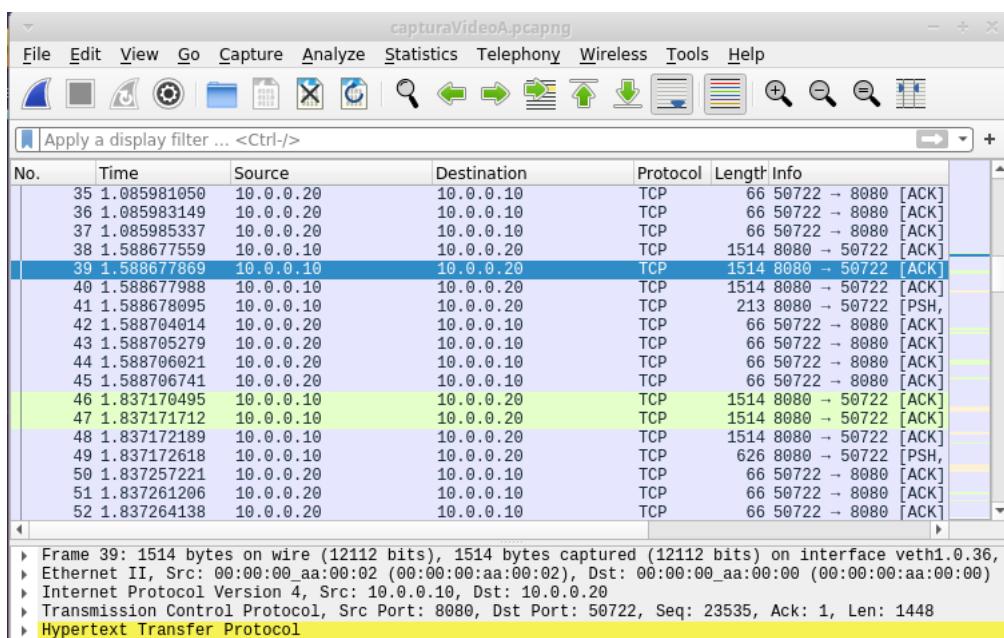


Figura 7: Encapsulamento protocolar com 1 cliente.

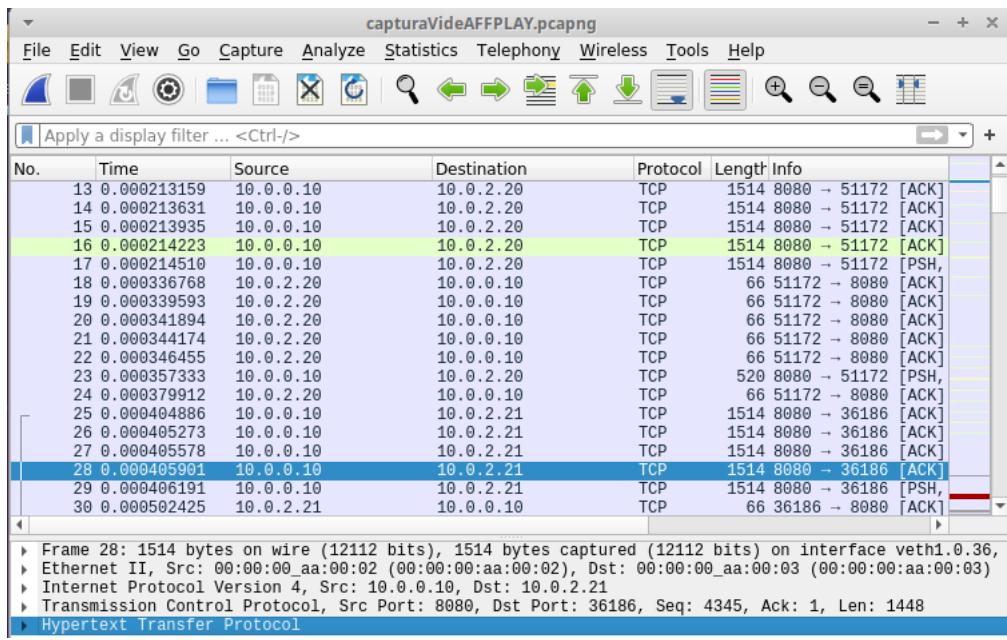


Figura 8: Encapsulamento protocolar com 2 clientes.

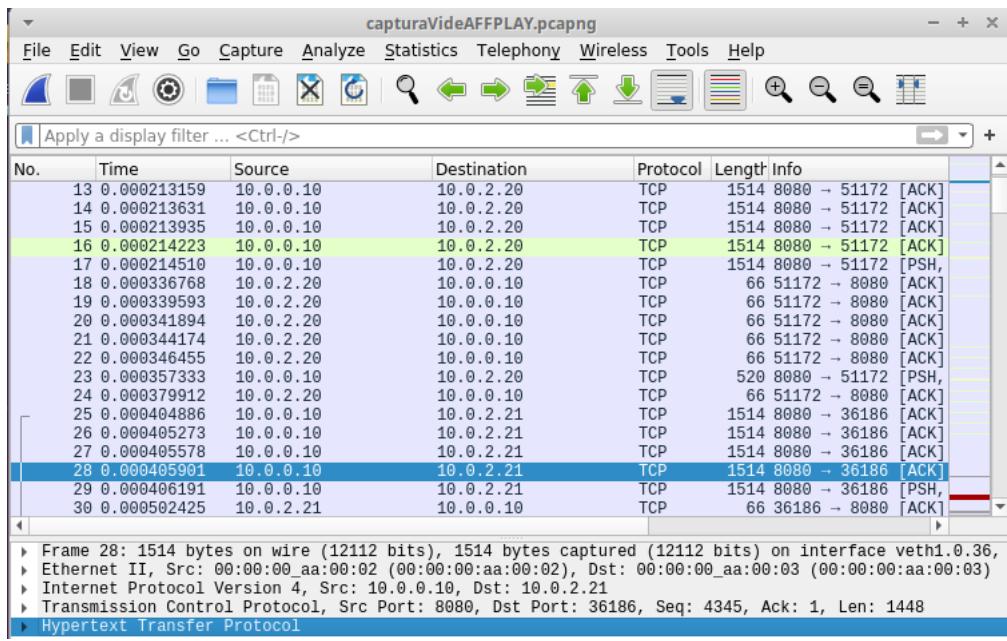


Figura 9: Encapsulamento protocolar com 3 clientes.

Acompanhando a Tabela acima, há encapsulamento ao nível das várias camadas da pilha, nomeadamente, camada de aplicação (HTTP), camada de transporte (TCP), camada de rede (IPv4), e na camada de interface de rede (Ethernet), mais propriamente na camada de ligação de dados, no modelo OSI.

As figuras acima, também mostram o encapsulamento, já que cada protocolo surge listado em detalhe. O que se verifica é que o encapsulamento é exatamente o mesmo para os três casos (VLC, Firefox e ffplay). A construção destes pacotes ao longo de cada camada surge evidenciada:

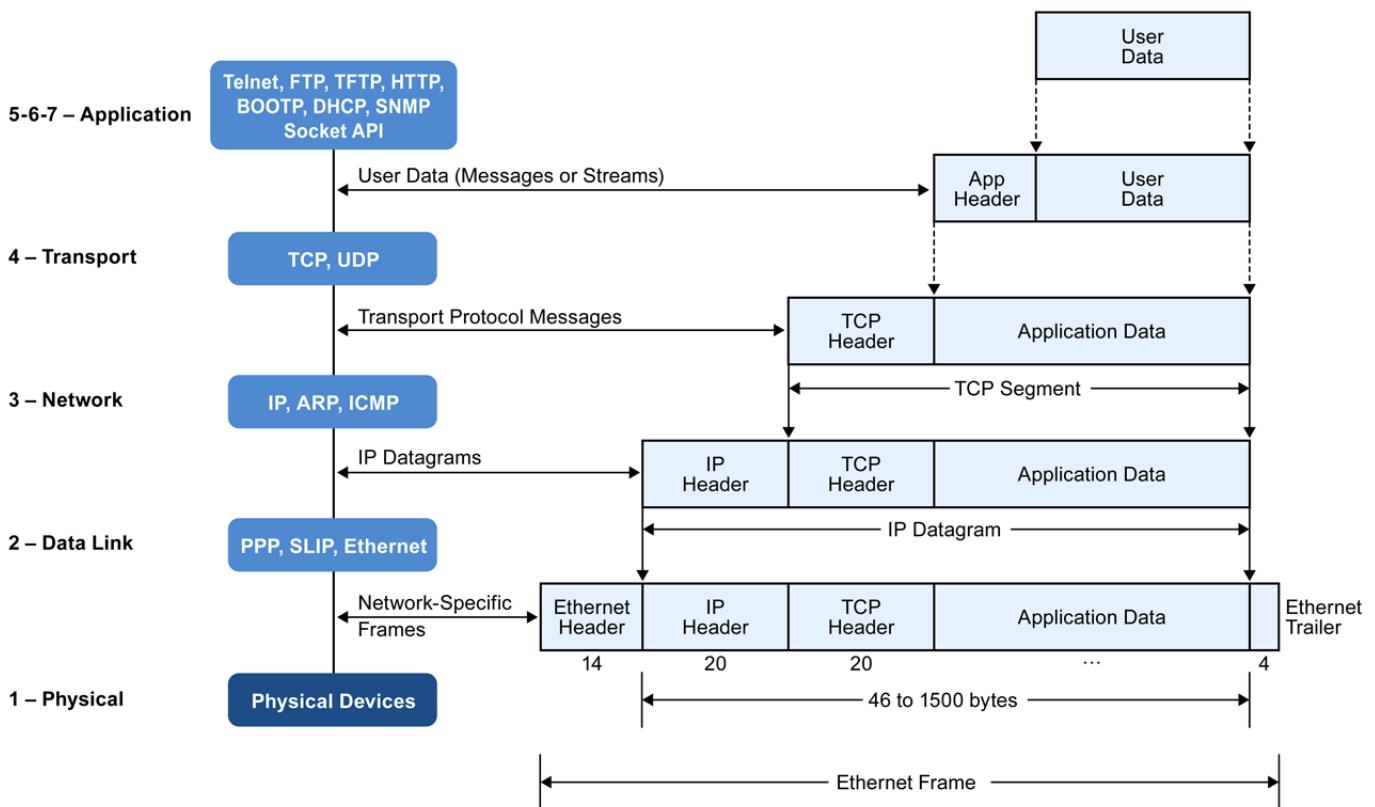


Figura 10: Encapsulamento protocolar nos pacotes (componentes)[5].

• Fluxos gerados:

Em relação ao número de fluxos gerados, podemos-nos socorrer de um comando do Wireshark: `tcp.stream eq <x>`, sendo x um número inteiro maior ou igual a zero. Este comando, atualmente, indica a ordem pela qual o programa detectou *TCP Streams*, então, a primeira verifica `tcp.stream == 0`, a próxima verifica `tcp.stream == 1`, e assim sucessivamente. Desta forma, é possível visualizar o número de fluxos gerados.

Em versões anteriores do programa, seria necessário especificar no filtro, um par de endereços IP e portas, o que resultava em resultados muito mais longos. Contudo, na nova versão, o comando `tcp.stream eq <x>` é suficiente. Mesmo assim, vamos suportar a nossa análise olhando para os esses campos: IP de Origem, IP de Destino, Porta de Origem, Porta de Destino.

Para maior facilidade, deixemos uma lista dos endereços ip de cada cliente envolvido (Jasmine, Bela & Monstro) e respetivo servidor de streaming:

- **Jasmine**: 10.0.0.20
- **Bela**: 10.0.2.20
- **Monstro**: 10.0.2.21
- **VStreamer**: 10.0.0.10

Então, começemos pelo caso em que temos apenas 1 cliente. Espera-se que exista apenas tráfego proveniente do VStreamer (10.0.0.10) para a Jasmine (10.0.0.20),

já que era o único cliente a transmitir no VLC. Como se observa abaixo, há tráfego a sair e a entrar na porta 8080 (VStreamer) para a porta que identifica a Jasmine (50722). Efetuando os comandos `tcp.stream eq 0` e `tcp.stream eq 1`, para identificar a primeira e segunda streams detetadas pelo programa, reparamos que não há tráfego para a segunda, logo, podemos concluir a existência de apenas 1 fluxo gerado, e por isso, 1 só cliente.

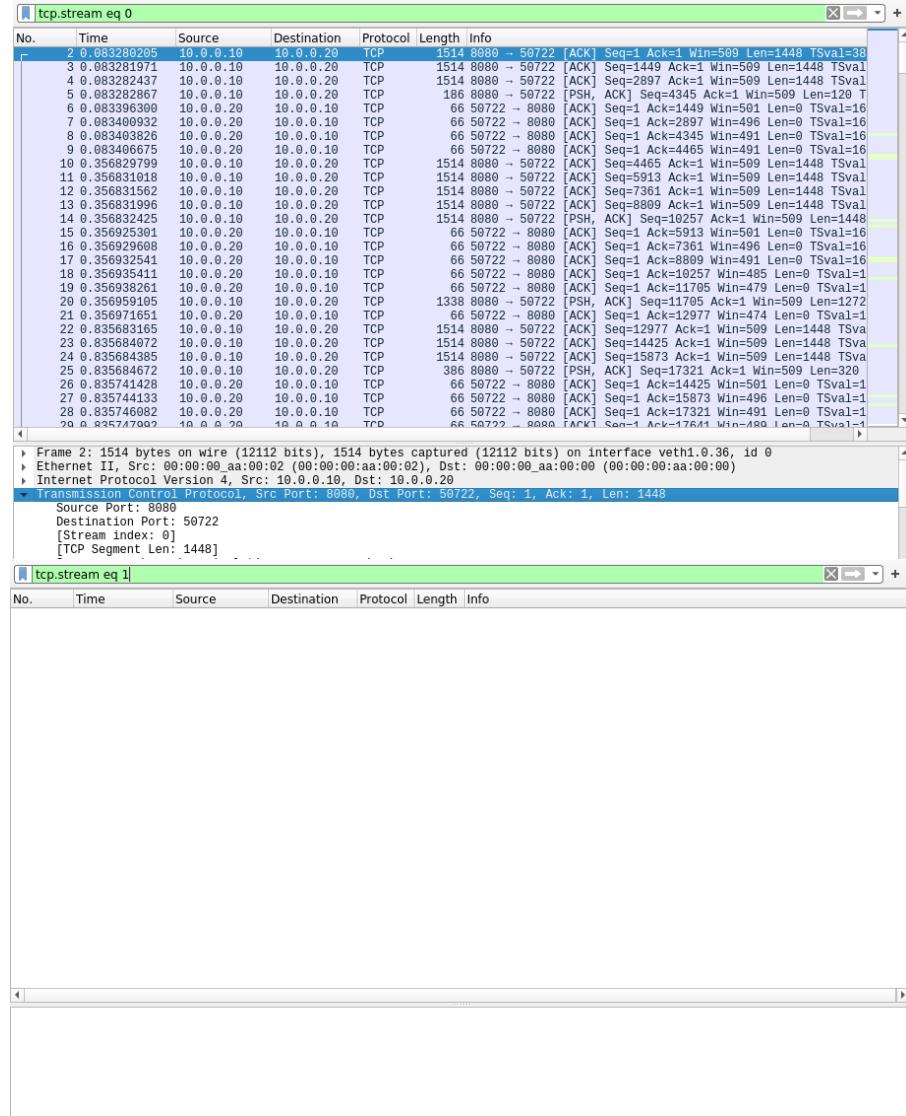


Figura 11: Fluxo gerado pelo cliente 1 - VStreamer.

No caso em que temos 2 clientes. Espera-se que exista tráfego entre o VStreamer (10.0.0.10) e Jasmine (10.0.0.20), no VLC, e a Bela (10.0.2.20), no Firefox. Efetuando os comandos `tcp.stream eq 0`, `tcp.stream eq 1` e `tcp.stream eq 2` para identificar a primeira, segunda e terceira streams detetadas pelo programa, reparamos que não há tráfego para a terceira, logo, podemos concluir a existência de apenas 2 fluxos gerados e, por isso, 2 clientes.

- `tcp.stream eq 0`: Há tráfego com origem em 10.0.0.20;
- `tcp.stream eq 1`: Há tráfego com origem em 10.0.2.20;

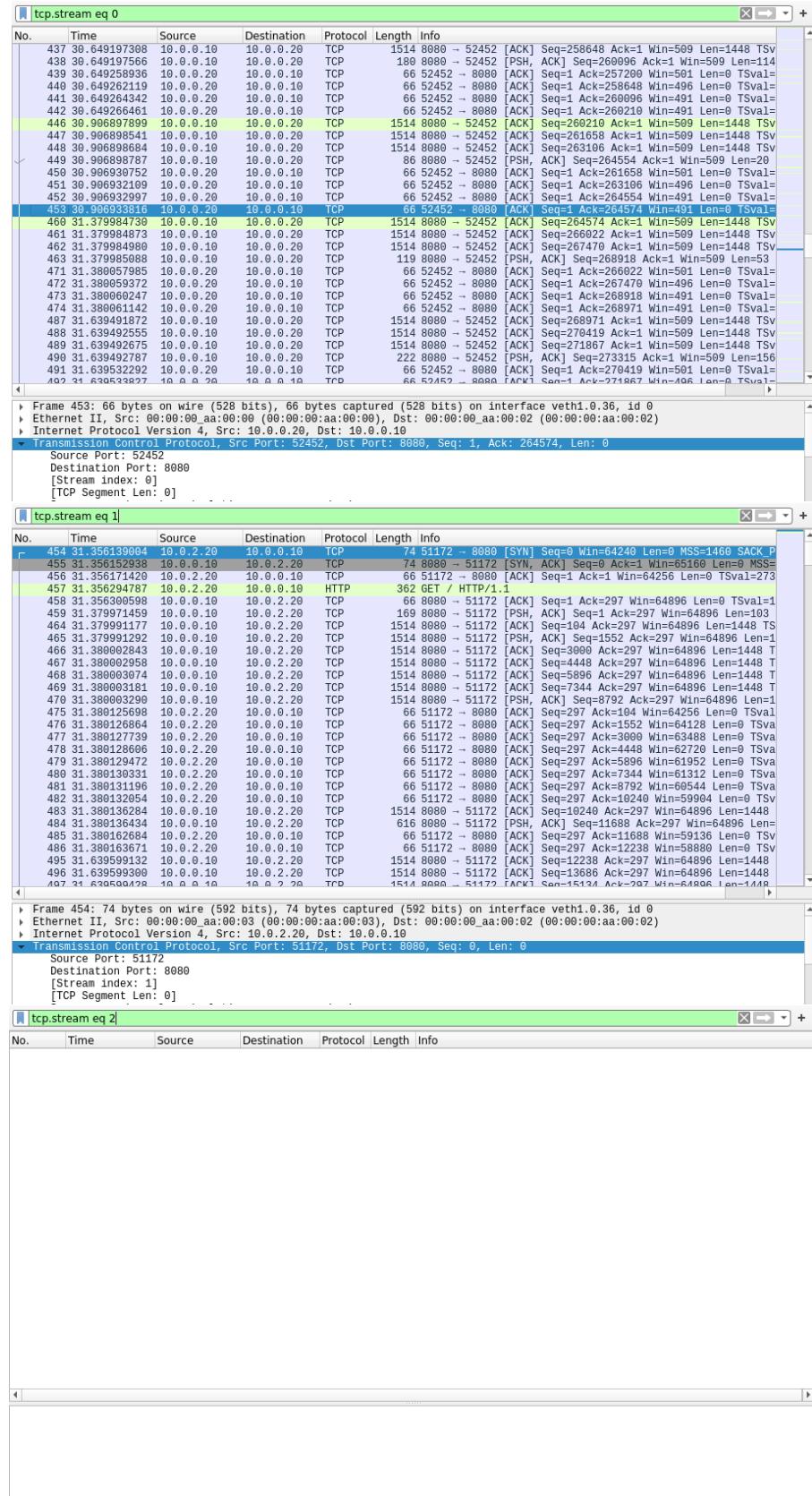


Figura 12: Fluxo gerado pelos 2 clientes - Jasmine.

Neste caso, em que há um cliente no Firefox, uma coisa interessante é ouvir a conversa entre o servidor e este cliente, seguindo o tráfego TCP, em cada stream. O cliente do Firefox foi o último a iniciar a conexão, então, com a *stream 1* selecionada, observamos a vermelha o que foi pedido por esse cliente e a resposta do servidor a azul. No canto inferior esquerdo, a existência de 2 clientes. No canto inferior direito, a existência das streams 0 e 1, correspondente a cada cliente.

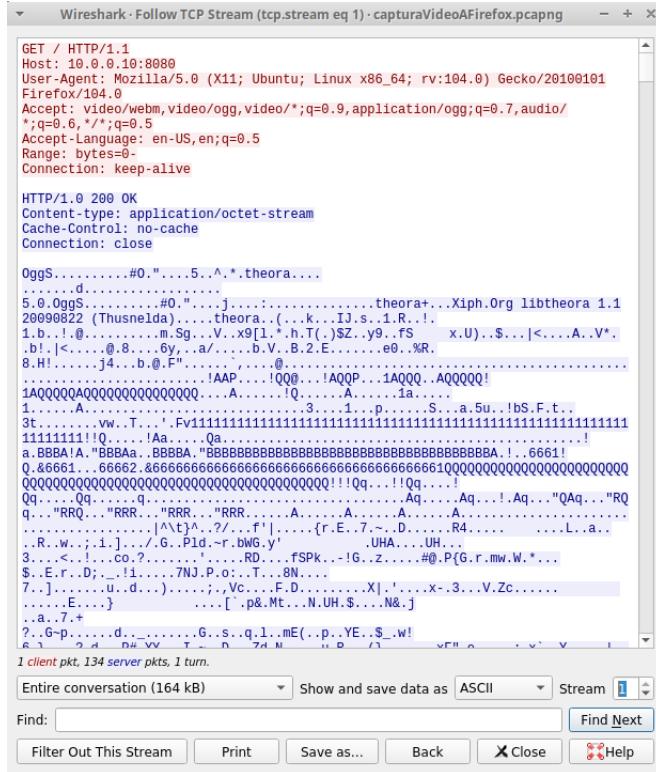


Figura 13: Tráfego entre os dois clientes - Jasmine e Bela.

No caso em que temos 3 clientes. Espera-se que exista tráfego do VStreame (10.0.0.10) com a Jasmine (10.0.0.20), no VLC, com a Bela, no Firefox e com o Monstro (10.0.2.21), com o ffplay.

Algo interessante e que o grupo só reparou depois desta análise das streams no Wireshark, é que, quando iniciamos a captura do programa, pensavamos que só existiam 3 clientes a funcionar (Jasmine no VLC, Bela no Firefox e Monstro no ffplay). Porém, ao analisar o número de streams, verificamos que existiam 4. Então, tinhhamos 1 cliente a mais.

Este cliente a mais, deve-se ao facto de, termos 2 separadores abertos no Firefox (dois clientes via Firefox). Assim sendo, afinal, existiam 4 clientes, 2 no Firefox, 1 no VLC e outro no ffplay.

- `tcp.stream eq 0:` Há tráfego com origem em 10.0.0.20;
- `tcp.stream eq 1:` Há tráfego com origem em 10.0.2.20;
- `tcp.stream eq 2:` Há tráfego com origem em 10.0.2.21;
- `tcp.stream eq 3:` Há tráfego com origem em 10.0.2.20;

tcp.stream eq 0										
No.	Time	Source	Destination	Protocol	Length	Info				
1	0.000000000	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=1 Win=509 Len=1448 Tsvl=38				
2	0.000001144	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
3	0.000001666	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=2897 Ack=1 Win=509 Len=1448 Tsvl=38				
4	0.000009288	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=345 Ack=1 Win=509 Len=1448 Tsvl=38				
5	0.0000092519	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [PSH, ACK] Seq=5793 Ack=1 Win=509 Len=1448 Tsvl=38				
6	0.000115940	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
8	0.000119551	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=5793 Win=631 Len=9 Tsvl=17				
9	0.000123137	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=5793 Win=626 Len=9 Tsvl=17				
10	0.000126577	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=7241 Win=620 Len=9 Tsvl=17				
11	0.000148428	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=5793 Win=620 Len=9 Tsvl=17				
12	0.000161524	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=7695 Win=618 Len=9 Tsvl=17				
13	0.000202192	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=7695 Win=618 Len=9 Tsvl=17				
39	0.273150678	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=8136 Win=647 Len=9 Tsvl=17				
44	1.273895989	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=8136 Ack=1 Win=509 Len=1448 Tsvl=17				
45	1.273896175	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=9584 Ack=1 Win=509 Len=1448 Tsvl=17				
46	1.273896275	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=11032 Ack=3 Win=509 Len=1448 Tsvl=17				
47	1.273896369	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=12482 Ack=3 Win=509 Len=1448 Tsvl=17				
48	1.273896428	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=13082 Ack=1 Win=509 Len=1448 Tsvl=17				
50	1.273913832	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=13928 Win=626 Len=9 Tsvl=1				
51	1.273914523	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=14376 Win=620 Len=9 Tsvl=1				
52	1.273915377	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=14928 Win=620 Len=9 Tsvl=1				
53	1.273916119	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=15376 Ack=1 Win=509 Len=1448 Tsvl=1				
54	1.273920633	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=15376 Ack=1 Win=509 Len=1448 Tsvl=1				
55	1.273920636	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=16824 Ack=1 Win=509 Len=1448 Tsvl=1				
57	1.273920641	18.0.0.20	10.0.0.10	TCP	1514	8080 - 52452 [ACK] Seq=16824 Ack=1 Win=509 Len=1448 Tsvl=1				
Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface veth1.0.36, id 0										
Ethernet II, Src: 00:00:00:aa:00:00 (00:00:00:aa:00:00), Dst: 00:00:00:aa:00:02 (00:00:00:aa:00:02)										
Internet Protocol Version 4, Src: 10.0.0.10, Dst: 10.0.0.20										
Transmission Control Protocol, Src Port: 52452, Dst Port: 60800, Seq: 1, Ack: 1, Win: 7695, Len: 0										
Source Port: 52452										
Destination Port: 8080										
[Stream index: 0]										
[TCP Segment Len: 0]										
tcp.stream eq 1										
No.	Time	Source	Destination	Protocol	Length	Info				
13	0.000202192	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1 Ack=Win=509 Len=1448 Tsvl=27				
14	0.000202193	18.0.0.10	10.0.0.28	TCP	1514	8080 - 52452 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=27				
16	0.000214223	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=27				
17	0.000214510	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [PSH, ACK] Seq=5793 Ack=1 Win=509 Len=1448 Tsvl=27				
18	0.000336768	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1449 Ack=1 Win=841 Len=441 Tsvl=27				
19	0.000339593	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=2897 Win=635 Len=9 Tsvl=27				
20	0.000344124	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=2897 Win=635 Len=9 Tsvl=27				
21	0.000344174	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=2897 Win=634 Len=9 Tsvl=27				
22	0.000346455	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=7241 Win=818 Len=9 Tsvl=27				
23	0.000347533	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [PSH, ACK] Seq=7241 Ack=1 Win=507 Len=1448 Tsvl=27				
24	0.000379912	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=7695 Win=818 Len=9 Tsvl=27				
40	1.273245636	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [PSH, ACK] Seq=7695 Ack=1 Win=507 Len=1448 Tsvl=27				
41	1.273249974	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=2897 Win=635 Len=9 Tsvl=27				
55	1.273968269	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=2897 Win=634 Len=9 Tsvl=27				
60	1.273968368	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=11032 Win=507 Len=1448 Tsvl=27				
61	1.273968461	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=11032 Win=507 Len=1448 Tsvl=27				
62	1.273968556	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [PSH, ACK] Seq=1 Ack=11032 Ack=1 Win=507 Len=1448 Tsvl=27				
63	1.274014943	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=11032 Win=618 Len=9 Tsvl=27				
65	1.274014987	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=11032 Win=618 Len=9 Tsvl=27				
66	1.274009885	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=12480 Win=829 Len=9 Tsvl=27				
67	1.274010595	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=15376 Win=821 Len=9 Tsvl=27				
68	1.274014900	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=15376 Ack=1 Win=507 Len=1448 Tsvl=27				
69	1.274015025	18.0.0.10	10.0.0.28	TCP	1514	8080 - 51172 [ACK] Seq=16824 Ack=1 Win=507 Len=1448 Tsvl=27				
70	1.274027430	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [PSH, ACK] Seq=1 Ack=16824 Win=812 Len=9 Tsvl=27				
71	1.274028343	18.0.0.20	10.0.0.10	TCP	1514	8080 - 51172 [ACK] Seq=1 Ack=17754 Win=800 Len=9 Tsvl=27				
Frame 13: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface veth1.0.36, id 0										
Ethernet II, Src: 00:00:00:aa:00:02 (00:00:00:aa:00:02), Dst: 00:00:00:aa:00:03 (00:00:00:aa:00:03)										
Internet Protocol Version 4, Src: 10.0.0.10, Dst: 10.0.0.20										
Transmission Control Protocol, Src Port: 8080, Dst Port: 51172, Seq: 1, Ack: 1, Len: 1448										
Source Port: 8080										
Destination Port: 51172										
[Stream index: 1]										
[TCP Segment Len: 1448]										
tcp.stream eq 2										
No.	Time	Source	Destination	Protocol	Length	Info				
26	0.0004049486	10.0.0.21	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1 Ack=1 Win=509 Len=1448 Tsvl=38				
27	0.0004095273	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
27	0.0004095278	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
28	0.0004095279	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
28	0.222916881	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916890	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916892	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916893	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916894	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916895	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916896	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916897	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916898	10.0.0.10	10.0.0.21	TCP	1514	8080 - 36186 [ACK] Seq=1449 Ack=1 Win=509 Len=1448 Tsvl=38				
29	0.222916899	10.0.0.10	10.0.0.21	TCP	1514					

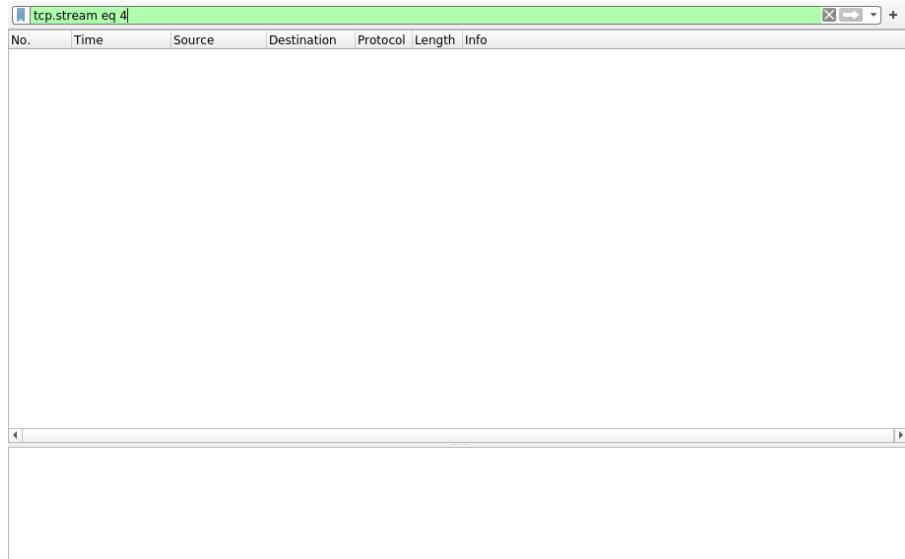


Figura 14: Fluxo gerado pelos 4 clientes - Jasmine, Bela (x2) e Monstro.

Desta forma, o que deveria ter acontecido seria existirem 3 fluxos gerados, devido à existência de 3 clientes por serviço. Contudo, no nosso caso, existindo 2 clientes no *Firefox*, registaram-se 4 clientes e, por isso, 4 fluxos gerados.

O grupo considerou que até foi bom isto ter acontecido, já que exploramos uma situação ligeiramente diferente e conseguimos observar melhor que, mesmo se tratando de um pedido semelhante (caso dos dois clientes no *Firefox*), o servidor envia as tramas individualmente para cada um.

Concluímos que existem tantos fluxos quanto o número de clientes existentes.

- **Escalabilidade:**

Antes de mais, é importante compreender o conceito de escala: a facilidade com que um sistema pode ser expandido, usando todos os seus recursos de forma a acomodar as exigências, sejam elas altas ou baixas.

Então, devemos perceber se esta arquitetura de Cliente(s)-Servidor, consegue facilmente crescer. Assim, podemos falar de escalabilidade horizontal - adição de mais recursos ao servidor - ou vertical - aumento do número de servidores ou clientes.

Por norma, uma arquitetura deste género é facilmente escalável, já que facilmente podemos aumentar o poder de resposta ao número crescente de utilizadores, aumentar o número de servidores ou os seus recursos.

Contudo, temos a desvantagem que se verificou de: se muitos clientes efetuarem pedidos ao mesmo servidor, este responderá individualmente a cada pedido, de modo sequencial, mesmo sendo o pedido o mesmo. Isto resulta numa maior congestão na rede e pode levar a problemas no acesso da informação, como se verificou no caso em que o vídeo no *Firefox* congelava, enquanto o servidor respondia ao pedido de outro cliente com o *ffplay*.

Concluindo, esta solução não é muito escalável, pois, perante o aumento do número de clientes, há uma resposta razoável em termos de performance.

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.

- **Largura de banda (bits/s):**

Podemos responder a esta questão através da consulta do ficheiro MPD produzido. Um ficheiro MPD (*Media Presentation Description*) é usado para armazenar as informações sobre os vários fluxos e as larguras de banda às quais eles estão associados.

Assim, é possível verificar a largura de banda dos diferentes vídeos através do conteúdo do ficheiro *video_manifest.mpd*, que guarda essa informação.

Após a sua análise, percebemos que a largura de banda necessária para que o cliente de streaming consiga receber o vídeo no *Firefox* é **114270 bits/s** no vídeo de menor resolução - $160 * 100$ (figura 15), **269849 bits/s** no vídeo de resolução intermédia - $320 * 200$ (figura16) e **668548 bits/s** no vídeo de maior resolução - $640 * 400$ (figura 17).

De seguida, apresentamos partes que consideramos importantes do conteúdo do ficheiro *video_manifest.mpd*.

```
</ProgramInformation>
<Period duration="PT0H0M12.4675">
<AdaptationSet segmentAlignment="true" bitstreamSwitching="true" maxWidth="640" maxHeight="400" maxFrameRate="30" sar="1:1" lang="und">
<SegmentList>
<Initialization sourceURL="video_manifest_init.mp4"/>
</SegmentList>
<Representation id="1" mimeType="video/mp4" codecs="avc3.64000c" width="160" height="100" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="114270">
<BaseURL>videoB_160_100_200k_dash.mp4</BaseURL>
<SegmentList timescale="15500" duration="7080">
<SegmentURL mediaRange="927-5915" indexRange="927-970"/>
<SegmentURL mediaRange="5916-9713" indexRange="5916-5959"/>
```

Figura 15: Largura de banda do vídeo de menor resolução ($160 * 100$)

```
</Representation>
<Representation id="2" mimeType="video/mp4" codecs="avc3.640014" width="320" height="200" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="269849">
<BaseURL>videoB_320_200_500k_dash.mp4</BaseURL>
<SegmentList timescale="15500" duration="7080">
<SegmentURL mediaRange="928-11993" indexRange="928-971"/>
<SegmentURL mediaRange="11994-21437" indexRange="11994-12037"/>
<SegmentURL mediaRange="21438-30539" indexRange="21438-21481"/>
<SegmentURL mediaRange="30540-52675" indexRange="30540-30583"/>
<SegmentURL mediaRange="52676-61239" indexRange="52676-52719"/>
<SegmentURL mediaRange="61240-67086" indexRange="61240-61283"/>
<SegmentURL mediaRange="67087-87519" indexRange="67087-67130"/>
<SegmentURL mediaRange="87520-102942" indexRange="87520-87563"/>
```

Figura 16: Largura de banda do vídeo de resolução intermédia ($320 * 200$)

```
</SegmentList>
</Representation>
<Representation id="3" mimeType="video/mp4" codecs="avc3.64001e" width="640" height="400" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="668548">
<BaseURL>videoB_640_400_1000k_dash.mp4</BaseURL>
<SegmentList timescale="15500" duration="7080">
<SegmentURL mediaRange="927-24612" indexRange="927-970"/>
<SegmentURL mediaRange="24613-47914" indexRange="24613-24656"/>
<SegmentURL mediaRange="47915-67271" indexRange="47915-47958"/>
<SegmentURL mediaRange="67272-115941" indexRange="67272-67315"/>
<SegmentURL mediaRange="115942-135163" indexRange="115942-115985"/>
<SegmentURL mediaRange="135164-150583" indexRange="135164-135207"/>
<SegmentURL mediaRange="150584-199813" indexRange="150584-150627"/>
```

Figura 17: Largura de banda do vídeo de maior resolução ($640 * 400$)

- **Pilha protocolar:**

A informação relativa à pilha protocolar usada no cenário de transmissão do vídeo encontra-se nas capturas do *Wireshark* (figuras 18 e 19), respetivas aos portáteis Bela e Alladin.

Como podemos observar, em ambos os casos os protocolos utilizados foram os mesmos:

- na camada de ligação de dados o protocolo **Ethernet**;
- na camada de rede o protocolo **IP**;
- na camada de transporte o protocolo **TCP**;
- na camada de aplicação o protocolo **HTTP**;

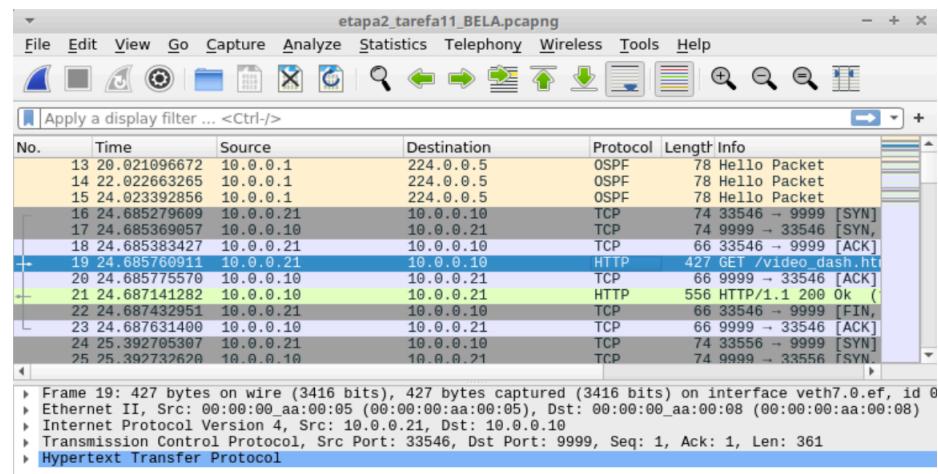


Figura 18: Pilha protocolar usada na transmissão do vídeo para o portátil Bela

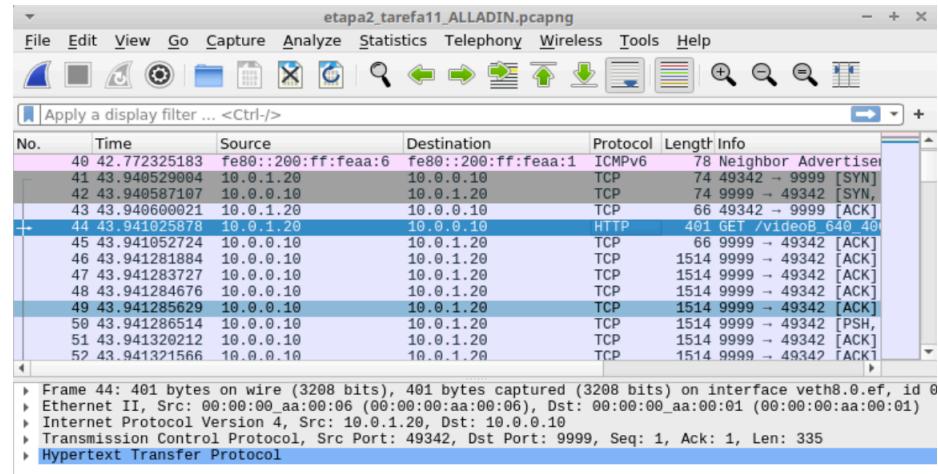


Figura 19: Pilha protocolar usada na transmissão do vídeo para o portátil Alladin

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 portátil Bela exiba o vídeo de menor resolução é necessário restringir o *link* da topologia para que esta permita a transmissão do vídeo de resolução 160×100 .

Como a largura de banda necessária à transmissão do vídeo de resolução baixa é 114270 bps, basta que o *link* do portátil Bela seja restringido à largura de banda de 200000 bps (200 kbps), conforme a figura 20.

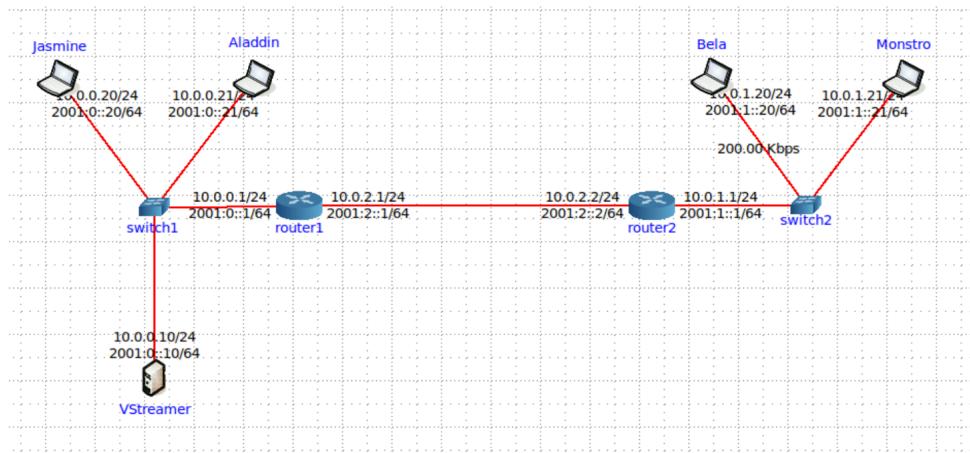


Figura 20: Topologia com alteração da restrição do link para o portátil Bela

Na captura do Wireshark para o portátil Bela (figura 21), podemos ver na linha 115 que há uma transmissão do vídeo de menor resolução, tal como era pretendido.

No.	Time	Source	Destination	Protocol	Length	Info
15	11.742659104	10.0.1.20	10.0.0.10	HTTP	381	GET /favicon.ico HTTP/1.1
17	11.775048570	10.0.0.10	10.0.1.20	HTTP	741	HTTP/1.1 404 Not Found (text/html)
115	15.925181718	10.0.1.20	10.0.0.10	HTTP	400	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
347	23.4390528210	10.0.0.10	10.0.1.20	MP4	238	
361	24.466889891	10.0.1.20	10.0.0.10	HTTP	401	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
611	31.947975755	10.0.0.10	10.0.1.20	MP4	238	
620	31.968174801	10.0.1.20	10.0.0.10	HTTP	402	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
876	39.438963844	10.0.0.10	10.0.1.20	MP4	238	
882	39.446158572	10.0.1.20	10.0.0.10	HTTP	404	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
902	39.709265198	10.0.1.20	10.0.0.10	HTTP	404	GET /videoB_160_100_200k_dash.mp4 HTTP/1.1
1166	47.907916927	10.0.0.10	10.0.1.20	MP4	238	

Figura 21: Captura da transmissão do vídeo para o portátil Bela

Para que portátil Alladin exiba o vídeo de maior resolução é necessário restringir o *link* da topologia para que esta permita a transmissão do vídeo de resolução 640×400 .

Como a largura de banda necessária à transmissão do vídeo de resolução alta é 668548 bps basta que o *link* do Alladin seja restringido à largura de banda de 7000000 bps (700 kbps) (figura 22).

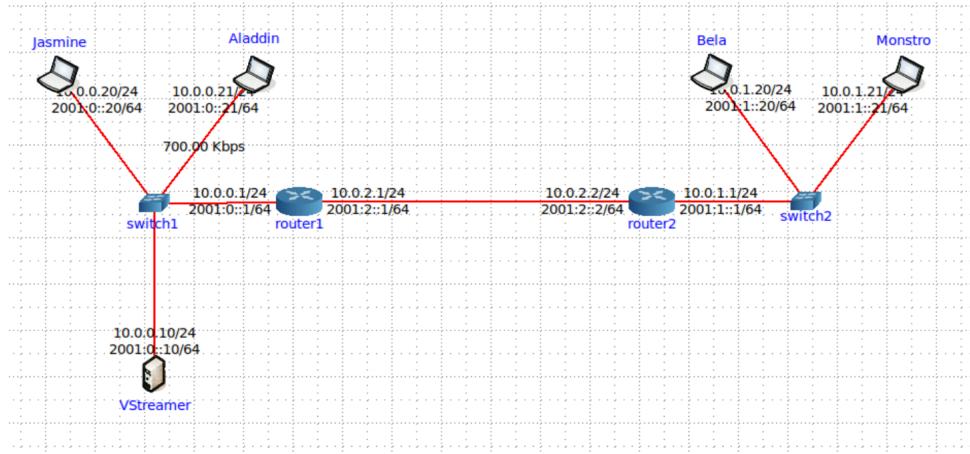


Figura 22: Topologia com alteração da restrição do link para o portátil Alladin

Na captura do Wireshark para o portátil Alladin (figura 23), podemos ver na linha 123 que há uma transmissão do vídeo de maior resolução (640^*400), tal como era pretendido.

Contudo, ao longo da transmissão, devido à funcionalidade do DASH, a resolução do vídeo vai diminuindo para a resolução intermédia (linha 977) e finalmente para a resolução mais baixa (linha 1463).

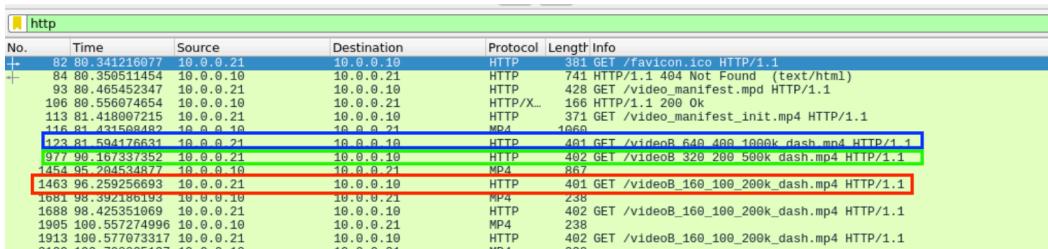


Figura 23: Captura da transmissão do vídeo para o portátil Alladin

- **Adaptação Dinâmica de Taxa:**

Perante a evidência anterior, frisemos esta característica de adaptação de taxa muito relevante. Esta capacidade de seleção dinâmica das taxas de bits de vídeo socorre-se de um conjunto de algoritmos, com vista a evitar interrupções na reprodução.

Esta capacidade é **acionada** assim que as condições de rede mudam. [6] O objetivo deste mecanismo é:

1. Maximizar a qualidade de vídeo;
2. Minimizar o número de trocas de qualidade de vídeo (melhorar a experiência do utilizador);
3. Minimizar o tempo de espera da requisição de um novo vídeo pelo utilizador e o tempo de espera até o vídeo estar pronto para ser reproduzido.

Nas capturas acima, conseguimos evidenciar esta adaptação através da diminuição para a resolução intermédia e, depois, para a mais baixa.

Todavia, esta mudança, em termos de experiência de utilizador, não deve ser (muito) evidente. O protocolo de transporte TCP tem um comportamento dinâmico que muitas vezes se torna imprevisível, gerando impacto na qualidade de

experiência do utilizador, mas estas tecnologias conseguem suavizar este problema.

Questão 4

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

- **Arquitetura DASH:**

Antes de explicar o funcionamento desta tecnologia, o grupo considerou relevante expor os aspectos principais da sua arquitetura, já que ela serve de suporte ao seu funcionamento.

Para tal, consideraremos o exemplo da Bela e do Alladin da questão anterior, apoiado pela figura seguinte:

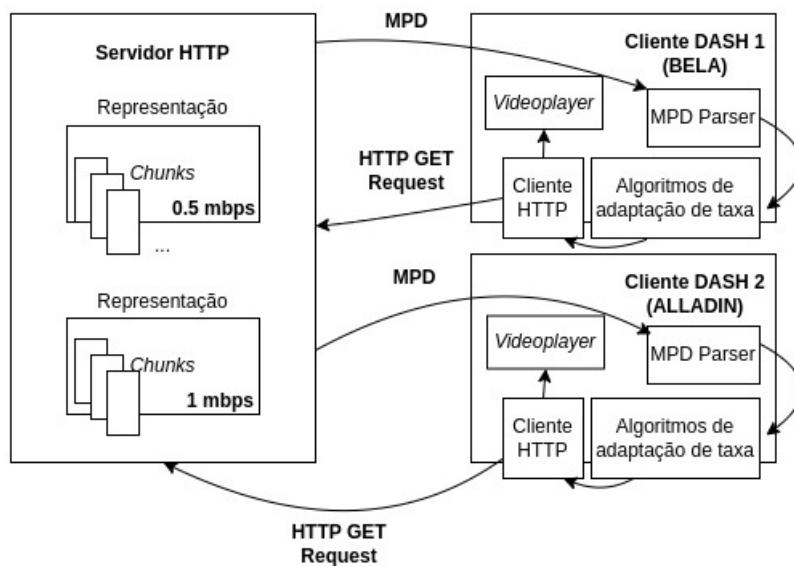


Figura 24: Arquitetura *DASH* e fluxo de acordo com o exemplo anterior (os valores das taxas são meramente exemplificativos).

O DASH começa por codificar o conteúdo do vídeo em várias versões com diferentes taxas de bits discretas (taxas de Representação), sendo o vídeo fragmentado em pequenos *chunks* do conteúdo original.

Então, forma-se uma linha temporal paralela com uma sequência de *chunks* (*profile*) das várias taxas, sendo isso o que permite ao cliente alternar suavemente entre as diferentes taxas de bits, se tiver necessidade.

Este *profile* é formado pelo tipo de codec de vídeo, áudio e resolução, por exemplo, e são estes elementos que vão determinar a taxa de bits desse mesmo *profile*.

Assim, a adaptação *streaming* de que falamos nada mais é do que a adequação que o servidor potencia ao cliente de acordo com a sua taxa de *download* disponível, ou seja, o pedido do cliente de acordo com a sua largura de banda, e a disponibilização do *profile* pelo servidor. [4]

- **Funcionamento DASH:**

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 DASH verifica a largura de banda do *link* e avalia qual a melhor resolução do vídeo a ser transmitida nesse *link*, a partir do ficheiro "video_manifest.mpd" (ou MPD), a base do seu funcionamento.

Durante a *stream* do vídeo, o DASH vai avaliando a qualidade de transmissão e a conectividade, alterando a resolução do vídeo conforme essas características da rede, como se evidenciou anteriormente.

Porém, este processo é detalhado:

1. Cliente faz um pedido *HTTP GET*, solicitando ao servidor o arquivo *MPD*, de modo a que o cliente obtenha informações sobre a taxa de bits do conteúdo, resolução, etc., de cada representação, bem como as URLs dos segmentos;
2. O cliente utiliza o algoritmo e decide qual perfil e segmento a fazer *download*;
3. O cliente vai reavaliando a situação: caso detete uma largura de banda baixa, sinaliza o servidor.

Desta forma, o grupo construiu o seguinte diagrama temporal que descreve este mesmo fluxo:

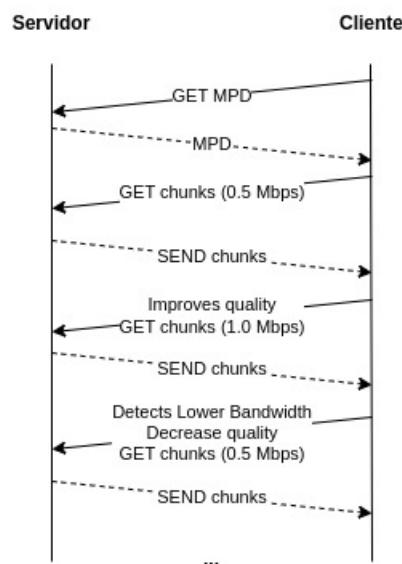


Figura 25: Funcionamento DASH e fluxo de acordo com o exemplo anterior (os valores das taxas são meramente exemplificativos).

Um exemplo do funcionamento do DASH ocorreu durante o *streaming* do vídeo no *firefox* no portátil Alladin.

Durante essa transmissão, foi realizada uma captura com o *Wireshark*, de forma a visualizar os dados recebidos pelo cliente.

Neste caso, analisamos que o *streaming* foi inicializado com o vídeo de resolução mais alta (640*400), contudo, ao longo da *stream*, verificamos que o vídeo

muda a sua resolução para a resolução intermédia ($320*200$) e, finalmente, para a resolução mais baixa ($160*100$), que acaba por ser a resolução transmitida até ao fim da *stream*.

Esta diminuição da resolução do vídeo ocorre, porque o *DASH* verifica que a conectividade e/ou a transmissão na rede não são suficientes para manter a transmissão integralmente a uma resolução mais alta ($640*400$) e, por isso, baixa a resolução do vídeo.

Ou seja, para que o utilizador consiga ver o vídeo completo sem que note perdas de *frame* ou pausas devido à falta de conexão, o *DASH* adapta a resolução do vídeo transmitido conforme as características da rede.

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

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.

Nesta etapa, o objetivo passava por testar em duas topologias de rede diferentes: o *streaming* em *unicast* (com apenas um *sender* a transmitir para um *receiver*) e em *multicast* (uma entidade a enviar para vários *hosts* na rede).

Relativamente ao serviço *multicast*, em termos de **escalabilidade**, é um serviço que é mais escalável do que o serviço *unicast*, uma vez que responde de uma maneira mais eficaz quando existem vários clientes na rede, aproveitando uma maior parte da largura de banda disponível e evitando enviar pacotes redundantes de informação.

Dito isto, o serviço *multicast* não só é melhor no que toca à escalabilidade, mas também no **tráfego de rede**, visto que se evita o envio de pacotes redundantes.

Contudo, em relação ao serviço *unicast*, o *multicast* tem uma complexidade mais elevada, o que faz com que este serviço tenha um maior *overhead* e mais custos de controlo.

Como é possível observar nas capturas de ecrã do Wireshark dos serviços *unicast* e *multicast*, a quantidade de dados no serviço *multicast* é ligeiramente superior, sendo esse um possível motivo para esta transmissão ter corrido durante mais tempo. Mas também os 175k correspondem a 4 clientes, enquanto os 138k do *unicast* correspondem a 1.

Logo, podemos comprovar que a transmissão *multicast* aproveita melhor a largura de banda e tem maior escalabilidade.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
Frame	100.0	1449	100.0	1178452	148 k	0
Ethernet	100.0	1449	1.7	20286	2555	0
Internet Protocol Version 6	0.4	6	0.0	240	30	0
Open Shortest Path First	0.4	6	0.0	216	27	6
Internet Protocol Version 4	99.3	1439	2.4	28780	3625	0
User Datagram Protocol	94.1	1363	0.9	10904	1373	0
Data	94.0	1362	93.1	1097400	138 k	1362
ADWin configuration protocol	0.1	1	0.0	52	6	1
Open Shortest Path First	2.2	32	0.1	1408	1	32
Internet Control Message Protocol	3.0	44	1.6	19054	2400	44
Address Resolution Protocol	0.3	4	0.0	112	14	4

Figura 26: Dados transmitidos pelo serviço *unicast*.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
Frame	100.0	1017	100.0	843825	185 k	0
Ethernet	100.0	1017	1.7	14238	3125	0
Internet Protocol Version 4	100.0	1017	2.4	20404	4478	0
User Datagram Protocol	98.4	1001	0.9	8008	1757	0
Session Announcement Protocol	0.8	8	0.3	2592	568	0
Session Description Protocol	0.8	8	0.3	2400	526	8
Real-Time Transport Protocol	96.9	985	94.6	798103	175 k	0
MP4V-ES	96.9	985	93.2	786283	175 k	985
Real-time Transport Control Protocol	0.8	8	0.0	224	4	8
Internet Group Management Protocol	1.6	16	0.0	256	56	16

Figura 27: Dados transmitidos pelo serviço *multicast*.

Com forma de tornar mais completa esta análise, observe-se a seguinte tabela, que procura evidenciar as diferenças entre cada tipo de *Streaming* [3][1][2]

Fator / Tipo	Unicast	Multicast
Intervenientes	1 remetente e 1 destinatário	Vários remetentes e vários destinatários
Fluxo de dados	Envio de dados de 1 dispositivo para apenas 1 outro dispositivo	Envio de dados de 1 dispositivo para vários outros
Contexto Topologia	Topologia Single-Node, o host contém toda a informação para a rede. Apenas um dispositivo conectado ao servidor, utilizado como centro de dados.	Funcionamento em topologias de outros tipos como: Star, Mesh, Tree e Hybrid
Escalabilidade	Não escala bem em streaming de dados.	Não escala bem em redes grandes.
Largura de Banda	Utiliza menos largura de banda. Há uma relação direta entre cada cliente e a capacidade do servidor. Cada cliente que se conecta ocupa largura de banda adicional.	Utilização eficiente. Não há uma relação direta entre o número de clientes e a capacidade do servidor. O envio de VMS (Video Management Streams) afeta toda a rede.
Mapeamento	De 1 para 1	De 1 para muitos

Tabela 2: Principais diferenças entre Unicast e Multicast, de acordo com parâmetros específicos.

Conclusão

Em jeito de conclusão, deixe-se o fluxo de tarefas evidente: começamos por gerar dois vídeos, que serviram de base para todas as etapas do trabalho.

Primeiramente, colocamos um servidor em *Streaming* para 1, 2 e 3 clientes, respectivamente, com diferentes ferramentas (VLC, Firefox e ffplay). Analisámos as várias taxas de transmissão e conseguimos concluir que estas taxas estão dependentes das características do meio (número de perdas) e número de clientes.

Segundamente, analisou a capacidade adaptativa do DASH na topologia e a sua arquitetura e funcionamento, compreendendo o papel fulcral do ficheiro manifesto em todo o procedimento.

Finalmente, compararamos os dois tipos de *Streaming* (RTP/RTCP unicast sobre UDP e multicast com anúncios SAP), compreendendo a escalabilidade e tráfego de rede de cada um.

O grupo considera que se cumpriram todos os requisitos solicitados e que este foi um projeto importante para consolidarmos os conhecimentos abordados nas aulas da disciplina de Engenharia de Serviços em Rede.

Referências

- [1] Difference between unicast and multicast | datavideo | professional end-to-end solutions provider for your live video production.
- [2] Rtp vs rtsp | explore rtp streaming why rtsp over tcp is better for video streaming online - contract engineering, product design development company - cardinal peak.
- [3] @adware (GeeksforGeeks). Difference between unicast and multicast, 10 2020.
- [4] Leandro Da and Silva De Melo. Streaming adaptativo sobre http.
- [5] Aniket Deshpande. Manets as propellant in the growth of the internet of things. *IOSR Journal of Computer Engineering*, 18:01-07, 2016.
- [6] Achraf Gazdar and Lamia Alkwai. Toward a full peer to peer mpeg-dash compliant streaming system. *Multimedia Tools and Applications*, 77:15829-15849, 6 2018.
- [7] James F Kurose, Keith W Ross, Boston Columbus, Indianapolis New, York San, Francisco Upper, Saddle River, Amsterdam Cape, Town Dubai, London Madrid, Milan Munich, Paris Montréal, Toronto Delhi, Mexico City São, Paulo Sydney, Hong Kong, Seoul Singapore, and Taipei Tokyo. Computer networking a top-down approach sixth edition. 2013.
- [8] Thomas Stockhammer. *Dynamic Adaptive Streaming over HTTP-Standards and Design Principles*. 2010.