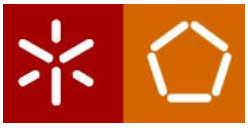


Universidade do Minho
Departamento de Informática

Engenharia de Serviços em Rede

Trabalho Prático 1

Grupo 86

Gonçalo Costa - PG55944 Lara Pereira - PG57884 Marta Rodrigues - PG55982

Índice

Questões e Respostas 3
Etapa 1. Streaming HTTP simples sem adaptação dinâmica de débito 3
Questão 1 3
Etapa 2. Streaming adaptativo sobre HTTP (MPEG-DASH) 6
Questão 2 6
Questão 3 8
Questão 4 9
Questão 5 10
Etapa 3. Streaming RTP/RTCP unicast sobre UDP e multicast com anúncios SAP 11
Questão 6 11
Conclusões 13

Questões e Respostas

Etapa 1. Streaming HTTP simples sem adaptação dinâmica de débito

Questão 1

Capture três pequenas amostras de trágefo no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (BVL 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).

Para a realização deste trabalho prático foi construida, como sugerido, a seguinte topologia.

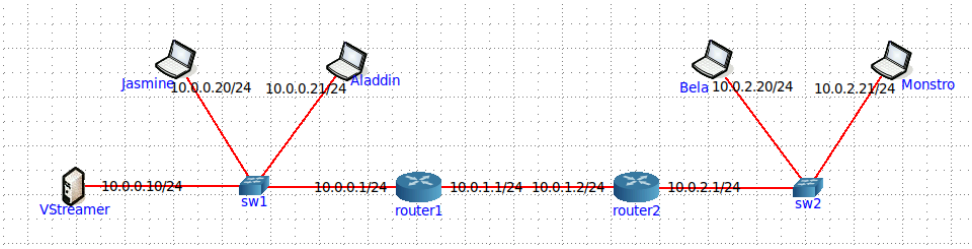


Figura 1: Topologia

Através do comando *ffmpeg -i videoA.mp4*, determinamos que a taxa em *bps* do vídeo A é 12kbps (12000 bps), como demonstra a figura a seguir.

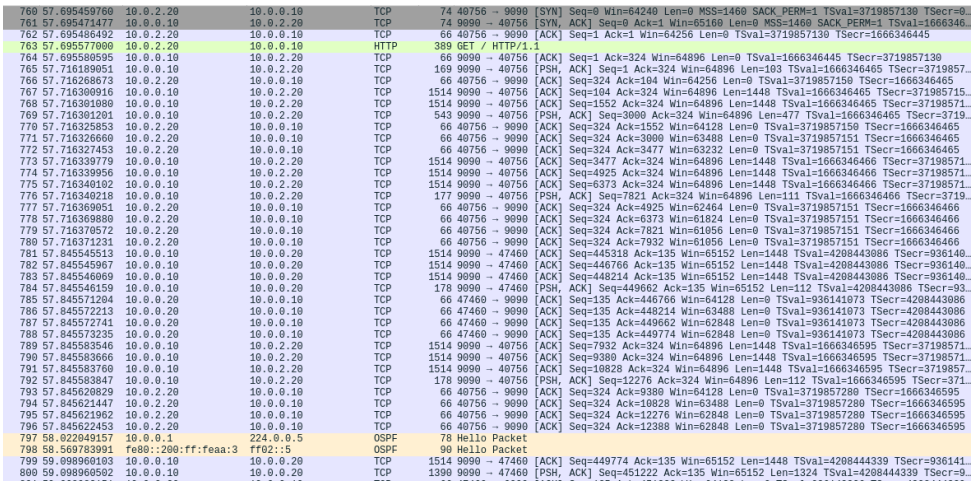
```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoA.mp4':
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder         : Lavf58.29.100
  Duration: 00:00:15.45, start: 0.000000, bitrate: 14 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 300x226, 12 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
    Metadata:
      handler_name    : VideoHandler
```

Figura 2: Captura do tráfego com *ffmpeg*

a) Comente os protocolos utilizados na transferência, bem como a experiência que o utilizador terá caso o link utilizado tenha perdas.

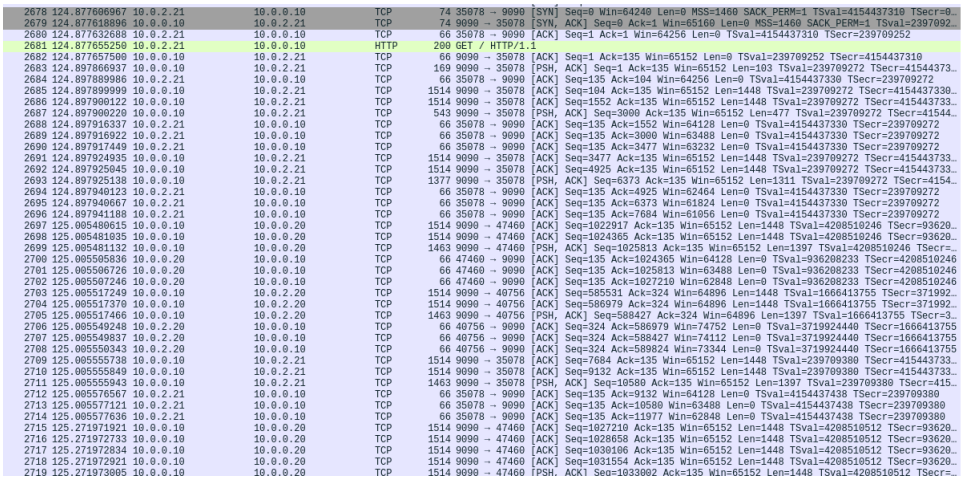
8	10.388004848	10.0.0.20	10.0.0.10	TCP	78 47460 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=936093615 TSecr=0 …
9	10.388019885	10.0.0.10	10.0.0.20	TCP	74 9090 → 47460 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=4208395…
10	10.388028454	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=936093615 TSecr=4208395628
11	10.388182166	10.0.0.20	10.0.0.10	HTTP	200 GET / HTTP/1.1
12	10.388186220	10.0.0.10	10.0.0.20	TCP	66 9090 → 47460 [ACK] Seq=1 Ack=135 Win=65152 Len=0 TSval=4208395628 TSecr=936093615
13	10.408379738	10.0.0.10	10.0.0.20	TCP	169 9090 → 47460 [PSH, ACK] Seq=1 Ack=135 Win=65152 Len=103 TSval=4208395649 TSecr=9360936…
14	10.408402073	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=104 Win=64256 Len=0 TSval=936093636 TSecr=4208395649
15	10.408417967	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=104 Ack=135 Win=65152 Len=1448 TSval=4208395649 TSecr=936093636…
16	10.408418106	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=1552 Ack=135 Win=65152 Len=1448 TSval=4208395649 TSecr=93609363…
17	10.408418220	10.0.0.10	10.0.0.20	TCP	543 9090 → 47460 [PSH, ACK] Seq=3000 Ack=135 Win=65152 Len=477 TSval=4208395649 TSecr=9360…
18	10.408450371	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=1552 Win=64128 Len=0 TSval=936093636 TSecr=4208395649
19	10.408451019	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=3000 Win=63488 Len=0 TSval=936093636 TSecr=4208395649
20	10.408451598	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=3477 Win=63232 Len=0 TSval=936093636 TSecr=4208395649
21	10.408487017	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=3477 Ack=135 Win=65152 Len=1448 TSval=4208395649 TSecr=93609363…
22	10.408487124	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=4925 Ack=135 Win=65152 Len=1448 TSval=4208395649 TSecr=93609363…
23	10.408487216	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=6373 Ack=135 Win=65152 Len=1448 TSval=4208395649 TSecr=93609363…
24	10.408487307	10.0.0.10	10.0.0.20	TCP	253 9090 → 47460 [PSH, ACK] Seq=7821 Ack=135 Win=65152 Len=187 TSval=4208395649 TSecr=9360…
25	10.408494584	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=4925 Win=62464 Len=0 TSval=936093636 TSecr=4208395649
26	10.408495150	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=6373 Win=61824 Len=0 TSval=936093636 TSecr=4208395649
27	10.408495658	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=7821 Win=61056 Len=0 TSval=936093636 TSecr=4208395649
28	10.408496148	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=8008 Win=60928 Len=0 TSval=936093636 TSecr=4208395649
29	10.570984042	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=8008 Ack=135 Win=65152 Len=1448 TSval=4208395811 TSecr=93609363…
30	10.570984657	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=9456 Ack=135 Win=65152 Len=1448 TSval=4208395811 TSecr=93609363…
31	10.570984936	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=10904 Ack=135 Win=65152 Len=1448 TSval=4208395811 TSecr=9360936…
32	10.570985191	10.0.0.10	10.0.0.20	TCP	91 9090 → 47460 [PSH, ACK] Seq=12352 Ack=135 Win=65152 Len=25 TSval=4208395811 TSecr=9360…
33	10.571033833	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=9456 Win=64128 Len=0 TSval=936093798 TSecr=4208395811
34	10.571036153	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=10904 Win=63488 Len=0 TSval=936093798 TSecr=4208395811
35	10.571037688	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=12352 Win=62848 Len=0 TSval=936093798 TSecr=4208395811
36	10.571039192	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=12377 Win=62848 Len=0 TSval=936093798 TSecr=4208395811
37	11.059811543	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=12377 Ack=135 Win=65152 Len=1448 TSval=4208396300 TSecr=9360937…
38	11.059812003	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=13825 Ack=135 Win=65152 Len=1448 TSval=4208396300 TSecr=9360937…
39	11.059812168	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=15273 Ack=135 Win=65152 Len=1448 TSval=4208396300 TSecr=9360937…
40	11.059812322	10.0.0.10	10.0.0.20	TCP	215 9090 → 47460 [PSH, ACK] Seq=16721 Ack=135 Win=65152 Len=149 TSval=4208396300 TSecr=936…
41	11.059846470	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=13825 Win=64128 Len=0 TSval=936094287 TSecr=4208396300
42	11.059847807	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=15273 Win=63488 Len=0 TSval=936094287 TSecr=4208396300
43	11.059848817	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=16721 Win=62848 Len=0 TSval=936094287 TSecr=4208396300
44	11.059849699	10.0.0.20	10.0.0.10	TCP	66 47460 → 9090 [ACK] Seq=135 Ack=16870 Win=62720 Len=0 TSval=936094287 TSecr=4208396300
45	11.546455207	10.0.0.10	10.0.0.20	TCP	1514 9090 → 47460 [ACK] Seq=16870 Ack=135 Win=65152 Len=1448 TSval=4208396787 TSecr=9360942…

Figura 3: Captura do tráfego com 1 cliente (VLC)



760 57.695450760	10.0.2.20	10.0.0.10	TCP	74	40756 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3719857130 TSecr=0…
761 57.695471477	10.0.0.10	10.0.2.20	TCP	74	9090 → 40756 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1666346…
762 57.695486492	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3719857130 TSecr=1666346445
763 57.695577000	10.0.2.20	10.0.0.10	HTTP	389	GET / HTTP/1.1
764 57.695580595	10.0.0.10	10.0.2.20	TCP	66	9090 → 40756 [ACK] Seq=1 Ack=324 Win=64896 Len=0 TSval=1666346445 TSecr=3719857130
765 57.716189051	10.0.0.10	10.0.2.20	TCP	169	9090 → 40756 [PSH, ACK] Seq=1 Ack=324 Win=64896 Len=103 TSval=1666346465 TSecr=3719857…
766 57.716268673	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=104 Win=64256 Len=0 TSval=3719857150 TSecr=1666346465
767 57.716300916	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=104 Ack=324 Win=64896 Len=1448 TSval=1666346465 TSecr=371985715…
768 57.716301080	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=1552 Ack=324 Win=64896 Len=1448 TSval=1666346465 TSecr=37198571…
769 57.716301201	10.0.0.10	10.0.2.20	TCP	543	9090 → 40756 [PSH, ACK] Seq=3000 Ack=324 Win=64896 Len=477 TSval=1666346465 TSecr=3719…
770 57.716325853	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=1552 Win=64128 Len=0 TSval=3719857150 TSecr=1666346465
771 57.716326660	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=3000 Win=63488 Len=0 TSval=3719857151 TSecr=1666346465
772 57.716327453	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=3477 Win=63232 Len=0 TSval=3719857151 TSecr=1666346465
773 57.716339779	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=3477 Ack=324 Win=64896 Len=1448 TSval=1666346466 TSecr=37198571…
774 57.716339956	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=4925 Ack=324 Win=64896 Len=1448 TSval=1666346466 TSecr=37198571…
775 57.716340102	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=6373 Ack=324 Win=64896 Len=1448 TSval=1666346466 TSecr=37198571…
776 57.716340218	10.0.0.10	10.0.2.20	TCP	177	9090 → 40756 [PSH, ACK] Seq=7821 Ack=324 Win=64896 Len=111 TSval=1666346466 TSecr=3719…
777 57.716369051	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=4925 Win=62464 Len=0 TSval=3719857151 TSecr=1666346466
778 57.716369880	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=6373 Win=61824 Len=0 TSval=3719857151 TSecr=1666346466
779 57.716370572	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=7821 Win=61056 Len=0 TSval=3719857151 TSecr=1666346466
780 57.716371231	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=7932 Win=61056 Len=0 TSval=3719857151 TSecr=1666346466
781 57.845545513	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=445318 Ack=135 Win=65152 Len=1448 TSval=4208443086 TSecr=936140…
782 57.845545967	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=446766 Ack=135 Win=65152 Len=1448 TSval=4208443086 TSecr=936140…
783 57.845546069	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=448214 Ack=135 Win=65152 Len=1448 TSval=4208443086 TSecr=936140…
784 57.845546159	10.0.0.10	10.0.0.20	TCP	178	9090 → 47460 [PSH, ACK] Seq=449662 Ack=135 Win=65152 Len=112 TSval=4208443086 TSecr=93…
785 57.845571204	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=446766 Win=64128 Len=0 TSval=936141073 TSecr=4208443086
786 57.845572213	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=448214 Win=63488 Len=0 TSval=936141073 TSecr=4208443086
787 57.845572741	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=449662 Win=62848 Len=0 TSval=936141073 TSecr=4208443086
788 57.845573235	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=449774 Win=62848 Len=0 TSval=936141073 TSecr=4208443086
789 57.845583546	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=7932 Ack=324 Win=64896 Len=1448 TSval=1666346595 TSecr=37198571…
790 57.845583666	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=9380 Ack=324 Win=64896 Len=1448 TSval=1666346595 TSecr=37198571…
791 57.845583760	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=10828 Ack=324 Win=64896 Len=1448 TSval=1666346595 TSecr=3719857…
792 57.845583847	10.0.0.10	10.0.2.20	TCP	178	9090 → 40756 [PSH, ACK] Seq=12276 Ack=324 Win=64896 Len=112 TSval=1666346595 TSecr=371…
793 57.845620829	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=9380 Win=64128 Len=0 TSval=3719857280 TSecr=1666346595
794 57.845621447	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=10828 Win=63488 Len=0 TSval=3719857280 TSecr=1666346595
795 57.845621962	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=12276 Win=62848 Len=0 TSval=3719857280 TSecr=1666346595
796 57.845622453	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=12388 Win=62848 Len=0 TSval=3719857280 TSecr=1666346595
797 58.022049157	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
798 58.569783991	fe80::200:ff:feaa:3	ff02::5	OSPF	90	Hello Packet
799 59.098960103	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=449774 Ack=135 Win=65152 Len=1448 TSval=4208444339 TSecr=936141…
800 59.098960502	10.0.0.10	10.0.0.20	TCP	1390	9090 → 47460 [PSH, ACK] Seq=451222 Ack=135 Win=65152 Len=1324 TSval=4208444339 TSecr=9…

Figura 4: Captura do tráfego com 2 clientes (VLC e Firefox)



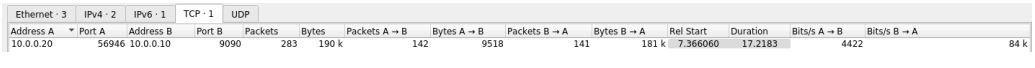
2678 124.877696967	10.0.2.21	10.0.0.10	TCP	74	35078 → 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=4154437310 TSecr=0…
2679 124.877618896	10.0.0.10	10.0.2.21	TCP	74	9090 → 35078 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2397092…
2680 124.877632688	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=4154437310 TSecr=23709252
2681 124.877655250	10.0.2.21	10.0.0.10	HTTP	200	GET / HTTP/1.1
2682 124.877657500	10.0.0.10	10.0.2.21	TCP	66	9090 → 35078 [ACK] Seq=1 Ack=135 Win=65152 Len=0 TSval=239709252 TSecr=4154437310
2683 124.897866937	10.0.0.10	10.0.2.21	TCP	169	9090 → 35078 [PSH, ACK] Seq=1 Ack=135 Win=65152 Len=103 TSval=239709272 TSecr=41544373…
2684 124.897889986	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=104 Win=64256 Len=0 TSval=4154437330 TSecr=239709272
2685 124.897899999	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=104 Ack=135 Win=65152 Len=1448 TSval=239709272 TSecr=415443733…
2686 124.897900122	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=1552 Ack=135 Win=65152 Len=1448 TSval=239709272 TSecr=415443733…
2687 124.897900220	10.0.0.10	10.0.2.21	TCP	543	9090 → 35078 [PSH, ACK] Seq=3000 Ack=135 Win=65152 Len=477 TSval=239709272 TSecr=41544…
2688 124.897916337	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=1552 Win=64128 Len=0 TSval=4154437330 TSecr=239709272
2689 124.897916922	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=3000 Win=63488 Len=0 TSval=4154437330 TSecr=239709272
2690 124.897917449	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=3477 Win=63232 Len=0 TSval=4154437330 TSecr=239709272
2691 124.897924935	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=3477 Ack=135 Win=65152 Len=1448 TSval=239709272 TSecr=415443733…
2692 124.897925045	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=4925 Ack=135 Win=65152 Len=1448 TSval=239709272 TSecr=415443733…
2693 124.897925138	10.0.0.10	10.0.2.21	TCP	1377	9090 → 35078 [PSH, ACK] Seq=6373 Ack=135 Win=65152 Len=1311 TSval=239709272 TSecr=4154…
2694 124.897940123	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=4925 Win=62464 Len=0 TSval=4154437330 TSecr=239709272
2695 124.897940667	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=6373 Win=61824 Len=0 TSval=4154437330 TSecr=239709272
2696 124.897941188	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=7684 Win=61056 Len=0 TSval=4154437330 TSecr=239709272
2697 125.005480615	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1022917 Ack=135 Win=65152 Len=1448 TSval=4208510246 TSecr=93620…
2698 125.005481035	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1024365 Ack=135 Win=65152 Len=1448 TSval=4208510246 TSecr=93620…
2699 125.005481132	10.0.0.10	10.0.0.20	TCP	1463	9090 → 47460 [PSH, ACK] Seq=1025813 Ack=135 Win=65152 Len=1397 TSval=4208510246 TSecr=…
2700 125.005505836	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=1024365 Win=64128 Len=0 TSval=936208233 TSecr=4208510246
2701 125.005506726	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=1025813 Win=63488 Len=0 TSval=936208233 TSecr=4208510246
2702 125.005507246	10.0.0.20	10.0.0.10	TCP	66	47460 → 9090 [ACK] Seq=135 Ack=1027210 Win=62848 Len=0 TSval=936208233 TSecr=4208510246
2703 125.005517249	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=585531 Ack=324 Win=64896 Len=1448 TSval=1666413755 TSecr=371992…
2704 125.005517370	10.0.0.10	10.0.2.20	TCP	1514	9090 → 40756 [ACK] Seq=586979 Ack=324 Win=64896 Len=1448 TSval=1666413755 TSecr=371992…
2705 125.005517466	10.0.0.10	10.0.2.20	TCP	1463	9090 → 40756 [PSH, ACK] Seq=588427 Ack=324 Win=64896 Len=1397 TSval=1666413755 TSecr=3…
2706 125.005549248	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=586979 Win=74752 Len=0 TSval=3719924440 TSecr=1666413755
2707 125.005549837	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=588427 Win=74112 Len=0 TSval=3719924440 TSecr=1666413755
2708 125.005550343	10.0.2.20	10.0.0.10	TCP	66	40756 → 9090 [ACK] Seq=324 Ack=589824 Win=73344 Len=0 TSval=3719924440 TSecr=1666413755
2709 125.005555738	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=7684 Ack=135 Win=65152 Len=1448 TSval=239709380 TSecr=415443733…
2710 125.005555849	10.0.0.10	10.0.2.21	TCP	1514	9090 → 35078 [ACK] Seq=9132 Ack=135 Win=65152 Len=1448 TSval=239709380 TSecr=415443733…
2711 125.005555943	10.0.0.10	10.0.2.21	TCP	1463	9090 → 35078 [PSH, ACK] Seq=10580 Ack=135 Win=65152 Len=1397 TSval=239709380 TSecr=415…
2712 125.005576567	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=9132 Win=64128 Len=0 TSval=4154437438 TSecr=239709380
2713 125.005577121	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=10580 Win=63488 Len=0 TSval=4154437438 TSecr=239709380
2714 125.005577636	10.0.2.21	10.0.0.10	TCP	66	35078 → 9090 [ACK] Seq=135 Ack=11977 Win=62848 Len=0 TSval=4154437438 TSecr=239709380
2715 125.271971921	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1027210 Ack=135 Win=65152 Len=1448 TSval=4208510512 TSecr=93620…
2716 125.271972733	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1028658 Ack=135 Win=65152 Len=1448 TSval=4208510512 TSecr=93620…
2717 125.271972834	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1030106 Ack=135 Win=65152 Len=1448 TSval=4208510512 TSecr=93620…
2718 125.271972921	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [ACK] Seq=1031554 Ack=135 Win=65152 Len=1448 TSval=4208510512 TSecr=93620…
2719 125.271973005	10.0.0.10	10.0.0.20	TCP	1514	9090 → 47460 [PSH, ACK] Seq=1033002 Ack=135 Win=65152 Len=1448 TSval=4208510512 TSecr=…

Figura 5: Captura do tráfego com 3 clientes (VLC, Firefox e ffplay)

Através do análise do tráfego da rede no *Wireshark*, verificamos que os protocolos utilizados na transferência são o HTTP (*Hypertext Transfer Protocol*) na camada de aplicação e o TCP (*Transmission Control Protocol*) na camada de transporte. Caso o link utilizado para a transferência sofra perdas, o TCP lidaria com o problema retransmitindo os pacotes perdidos o que pode criar atrasos na transferência que podem ser percecionados pelo utilizador como lentidão e/ou interrupções na transmissão, e o HTTP pode, no caso de as retransmissões falharem, apresentar ao utilizador falhas a carregar o conteúdo da *stream* ou erros.

b) Identifique o número total de fluxos gerados e elabore um gráfico que demonstre a evolução do débito dependendo do número de clientes.

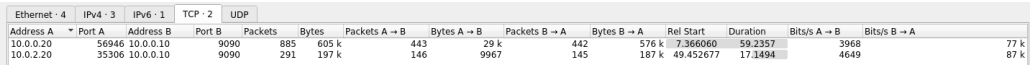
Analisando o tráfego gerado nos três instantes testados, isto é, um com 1 cliente, outro com 2 clientes e por último com 3 clientes, adquirimos os seguintes resultados.



Ethernet · 3 | IPv4 · 2 | IPv6 · 1 | TCP · 1 | UDP

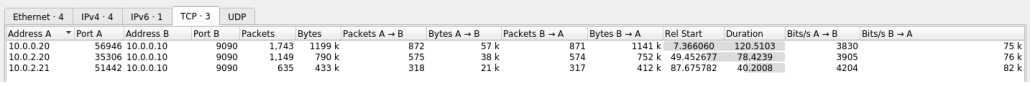
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	56946	10.0.0.10	9090	283	190 k	142	9518	141	181 k	7.366060	17.2183	4422	84 k

Figura 6: *Statistics* para 1 cliente (VLC)



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	56946	10.0.0.10	9090	885	605 k	443	29 k	442	576 k	7.366060	59.2357	3968	77 k
10.0.2.20	35306	10.0.0.10	9090	291	197 k	146	9967	145	187 k	49.452677	17.1494	4649	87 k

Figura 7: *Statistics* para 2 clientes (VLC e Firefox)

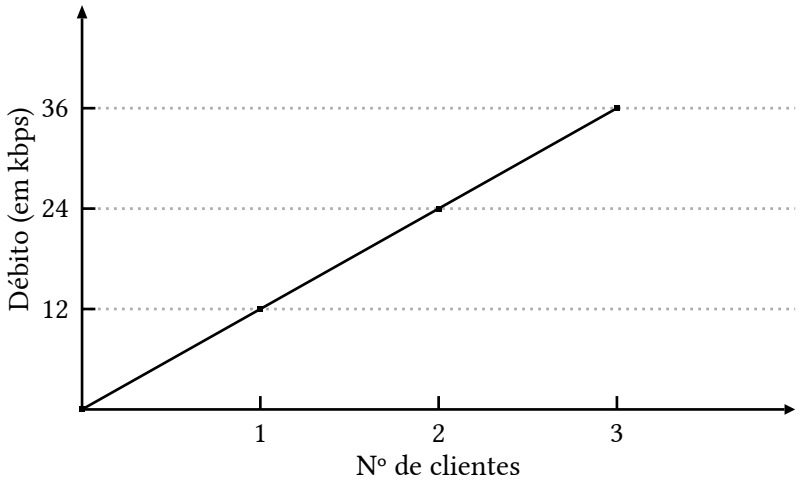


Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	56946	10.0.0.10	9090	1,743	1199 k	872	57 k	871	1141 k	7.366060	120.5103	3830	75 k
10.0.2.20	35306	10.0.0.10	9090	1,149	790 k	575	38 k	574	752 k	49.452677	78.4239	3905	76 k
10.0.2.21	51442	10.0.0.10	9090	635	433 k	318	21 k	317	412 k	87.675782	40.2008	4204	82 k

Figura 8: *Statistics* para 3 clientes (VLC, Firefox e ffplay)

Como é possível observar nas figuras, no momento que a Jasmine (IP: 10.0.0.20) se conecta à *stream*, pelo VLC, é gerado 1 fluxo. Já quando a Bela (IP: 10.0.2.20) também estabelece conexão através do *Firefox*, passamos a ter 2 fluxos. Finalmente, ao conectar o último cliente, o Monstro (IP: 10.0.2.21), temos 3 diferentes fluxos.

Estes dados permitem-nos concluir que o número de fluxos gerados na transferência dos dados para os múltiplos clientes corresponde ao número de clientes conectados ao servidor, ou seja, é criado um fluxo por cliente. Portanto, o débito necessário para fazer o *streaming* para os múltiplos clientes vai aumentar linearmente, pois, para cada cliente adicional, o servidor precisa enviar a mesma taxa de bits (bps) do vídeo. Podemos então afirmar que o débito total exigido pelo servidor é diretamente proporcional ao número de clientes, tal como é representado no gráfico seguinte, refletindo uma escalabilidade linear.



Graph 1: Evolução do débito dependendo do número de clientes

c) Comente a escalabilidade da solução para 1000 utilizadores, assim como 10000 utilizadores. Crie uma expressão matemática que expresse o débito necessário para que o servidor envie vídeo para N clientes.

Tal como foi observado na alínea anterior, o *streaming* por HTTP possui uma escalabilidade linear sendo que é gerado um novo fluxo de dados por cada novo cliente, por isso a cada novo cliente o débito necessário para o servidor enviar o vídeo aos clientes aumenta proporcionalmente. Tendo em conta estas observações, o débito necessário para transferir o vídeo para N clientes pode ser expresso através da seguinte fórmula, sendo B a taxa em bps necessária:

$$D(N) = N \times B$$

Logo, para casos com 1000 utilizadores o débito necessário para a transferência do vídeo A seria :

$$D(1000) = 1000 \times 12kbps = 12000kbps$$

E para 10000 utilizadores seria preciso :

$$D(10000) = 10000 \times 12kbps = 120000kbps$$

Estes valores são muito difíceis de alcançar pois requerem uma largura de banda significativa. Isto torna a escalabilidade linear obtida muito negativa, visto que quanto maior for o número de clientes, maior a largura de banda requerida do lado do servidor para atender a todos os pedidos dos vários utilizadores.

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

Questão 2

Utilize o wireshark para determinar 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. Explique como obteve esta informação

Ethernet · 3 | IPv4 · 2 | IPv6 · 1 | TCP · 6 | UDP

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	10.0.2.20	450	363 k	251	348 k	199	14 k	2.979341	17.1211	162 k	
10.0.2.1	224.0.0.5	17	1326	17	1326	0	0	0.000000	32.0175	331	

Figura 9: Captura de Statistics -> Conversations -> IPv4 do video com menor resolução

Ethernet · 4 | IPv4 · 2 | IPv6 · 4 | TCP · 5 | UDP

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	10.0.2.20	1,620	1598 k	1,040	1559 k	580	38 k	12.787996	11.7978	1057 k	26 k
10.0.2.1	224.0.0.5	19	1482	19	1482	0	0	0.000000	36.0550	328	0

Figura 10: Captura de Statistics -> Conversations -> IPv4 do video com maior resolução

A largura de banda necessária para o cliente receber o vídeo com melhor resolução é de 1057k bits/s.

Esse valor foi obtido acedendo a *Statistics -> Conversations -> IPv4* após observar o ficheiro *video_manifest.mpd* que, como podemos ver na figura abaixo, nos indica que o ficheiro de vídeo com resolução mais baixa possui uma *bandwidth* de 883116 bits/s, após manipular a *bandwith* do link entre o *switch* 2 e o cliente Bela escalando o valor observado a partir de 883116 bits/s. Testamos com vários valores como por exemplo 1M bits/s e 1.2M bits/s, eventualmente descobrindo que o valor mínimo para do link de ligação para o cliente receber o vídeo esperado foi por volta de 1.5M bits/s.

```
</Representation>
<Representation id="3" mimeType="video/mp4" codecs="avc3.64001f" width="960" height="720" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="883116">
 <BaseURL>videoB_960_720_1000k_dash.mp4</BaseURL>
 <SegmentList timescale="15360" duration="7680">
  <SegmentURL mediaRange="927-49578" indexRange="927-970"/>
```

Figura 11: Captura de *video_manifest.mpd*

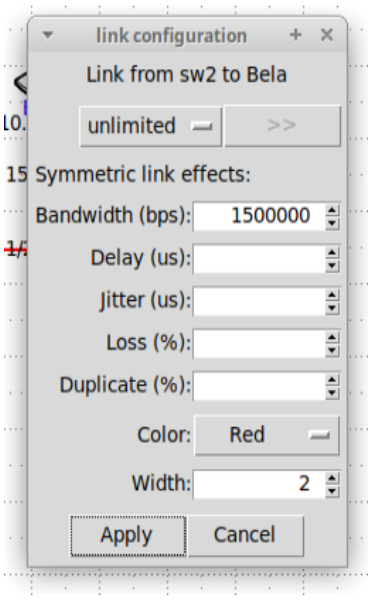


Figura 12: Captura da limitação do link

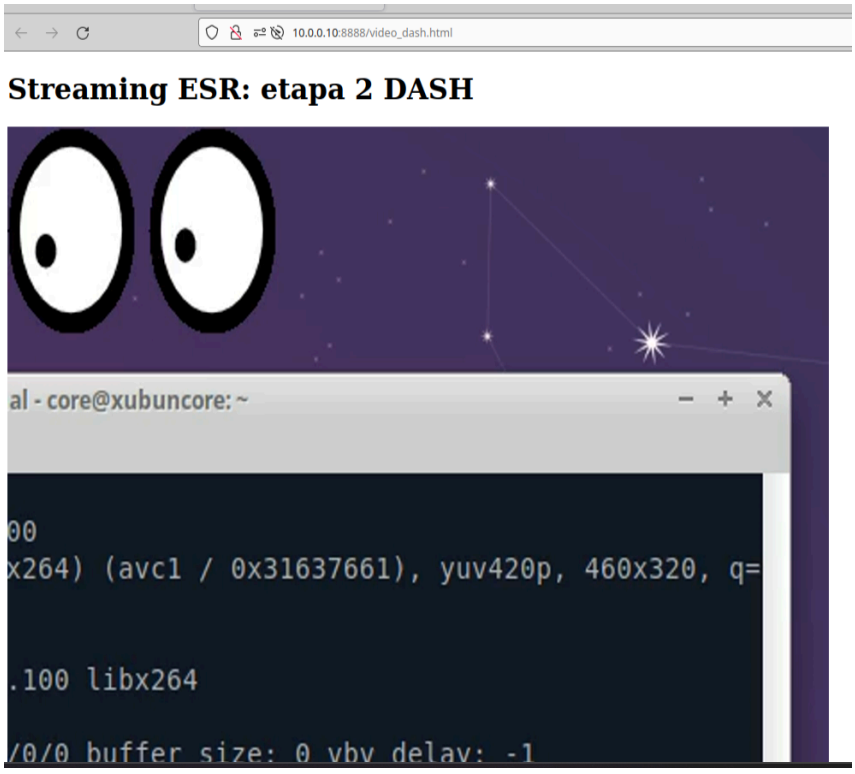
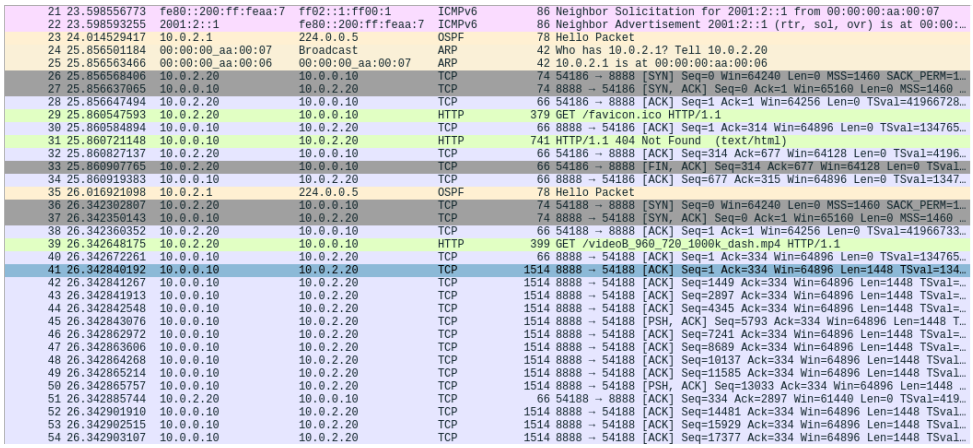


Figura 13: Visualização do Firefox do cliente com a maior resolução

Quanto à pilha protocolar podemos observar na captura que os pacotes usam o protocolo TCP, referente à camada de transporte. Pelos endereços IP, pode-se inferir o uso do protocolo IPv4 na camada de rede. Quanto a nível de aplicação podemos observar a utilização de HTTP.

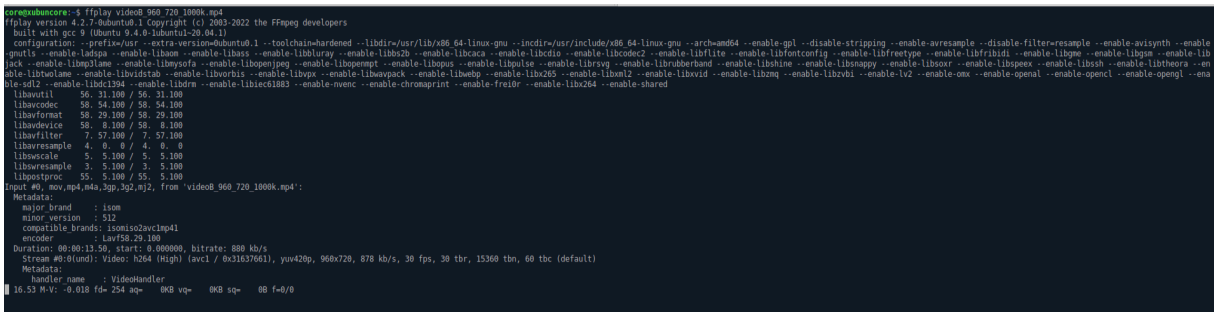


21	23.598556773	fe80::200:ff:feaa:7	ff02::1:ff00:1	ICMPv6	86 Neighbor Solicitation for 2001:2::1 from 00:00:00:aa:00:07
22	23.598593255	2001:2::1	fe80::200:ff:feaa:7	ICMPv6	86 Neighbor Advertisement 2001:2::1 (rtr, sol, ovr) is at 00:00:…
23	24.014529417	10.0.2.1	224.0.0.5	OSPF	78 Hello Packet
24	25.856501184	00:00:00_aa:00:07	Broadcast	ARP	42 Who has 10.0.2.1? Tell 10.0.2.20
25	25.856563466	00:00:00_aa:00:06	00:00:00_aa:00:07	ARP	42 10.0.2.1 is at 00:00:00:aa:00:06
26	25.856568406	10.0.2.20	10.0.0.10	TCP	74 54186 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
27	25.856637065	10.0.0.10	10.0.2.20	TCP	74 8888 → 54186 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
28	25.856647494	10.0.2.20	10.0.0.10	TCP	66 54186 → 8888 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=41966728…
29	25.860547593	10.0.2.20	10.0.0.10	HTTP	379 GET /favicon.ico HTTP/1.1
30	25.860584894	10.0.0.10	10.0.2.20	TCP	66 8888 → 54186 [ACK] Seq=1 Ack=314 Win=64896 Len=0 TSval=134765…
31	25.860721148	10.0.0.10	10.0.2.20	HTTP	741 HTTP/1.1 404 Not Found (text/html)
32	25.860827137	10.0.2.20	10.0.0.10	TCP	66 54186 → 8888 [ACK] Seq=314 Ack=677 Win=64128 Len=0 TSval=4196…
33	25.860907765	10.0.2.20	10.0.0.10	TCP	66 54186 → 8888 [FIN, ACK] Seq=314 Ack=677 Win=64128 Len=0 TSval…
34	25.860919383	10.0.0.10	10.0.2.20	TCP	66 8888 → 54186 [ACK] Seq=677 Ack=315 Win=64896 Len=0 TSval=1347…
35	26.016921098	10.0.2.1	224.0.0.5	OSPF	78 Hello Packet
36	26.342302807	10.0.2.20	10.0.0.10	TCP	74 54188 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
37	26.342350143	10.0.0.10	10.0.2.20	TCP	74 8888 → 54188 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
38	26.342360352	10.0.2.20	10.0.0.10	TCP	66 54188 → 8888 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=41966733…
39	26.342648175	10.0.2.20	10.0.0.10	HTTP	399 GET /videoB_960_720_1000k_dash.mp4 HTTP/1.1
40	26.342672261	10.0.0.10	10.0.2.20	TCP	66 8888 → 54188 [ACK] Seq=1 Ack=334 Win=64896 Len=0 TSval=134765…
41	26.342840192	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=1 Ack=334 Win=64896 Len=1448 TSval=134…
42	26.342841267	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=1449 Ack=334 Win=64896 Len=1448 TSval=…
43	26.342841913	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=2897 Ack=334 Win=64896 Len=1448 TSval=…
44	26.342842548	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=4345 Ack=334 Win=64896 Len=1448 TSval=…
45	26.342843076	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [PSH, ACK] Seq=5793 Ack=334 Win=64896 Len=1448 T…
46	26.342862972	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=7241 Ack=334 Win=64896 Len=1448 TSval=…
47	26.342863606	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=8689 Ack=334 Win=64896 Len=1448 TSval=…
48	26.342864268	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=10137 Ack=334 Win=64896 Len=1448 TSval…
49	26.342865214	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=11585 Ack=334 Win=64896 Len=1448 TSval…
50	26.342865757	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [PSH, ACK] Seq=13033 Ack=334 Win=64896 Len=1448 …
51	26.342885744	10.0.2.20	10.0.0.10	TCP	66 54188 → 8888 [ACK] Seq=334 Ack=2897 Win=61440 Len=0 TSval=419…
52	26.342901910	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=14481 Ack=334 Win=64896 Len=1448 TSval…
53	26.342902515	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=15929 Ack=334 Win=64896 Len=1448 TSval…
54	26.342903107	10.0.0.10	10.0.2.20	TCP	1514 8888 → 54188 [ACK] Seq=17377 Ack=334 Win=64896 Len=1448 TSval…

Figura 14: Captura do tráfego no cliente

Questão 3

Compare a largura de banda medida na questão anterior com a que é disponibilizada pelo ffplay. Qual é a razão para a diferença entre as duas?



```
core@xubuncore:~$ ffplay videoB_960_720_1000k.mp4
ffplay version 4.2.7-0ubuntu0.1 Copyright (c) 2003-2022 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.4.0-1ubuntu1~20.04.1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresample --disable-filter=resample --enable-avisynth --enable
-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libcodec2 --enable-libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-lib
jack --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librsvg --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libtheora --en
able-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-libzvbi --enable-lv2 --enable-omx --enable-openal --enable-opencl --enable-opengl --ena
ble-sdl2 --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-nvenc --enable-chromaprint --enable-frei0r --enable-libx264 --enable-shared
  libavutil      56. 31.100 / 56. 31.100
  libavcodec     58. 54.100 / 58. 54.100
  libavformat    58. 29.100 / 58. 29.100
  libavdevice    58.  8.100 / 58.  8.100
  libavfilter     7. 57.100 /  7. 57.100
  libavresample   4.  0.  0 /  4.  0.  0
  libswscale      5.  5.100 /  5.  5.100
  libswresample   3.  5.100 /  3.  5.100
  libpostproc    55.  5.100 / 55.  5.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoB_960_720_1000k.mp4':
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder         : Lavf58.29.100
  Duration: 00:00:13.50, start: 0.000000, bitrate: 880 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 960x720, 878 kb/s, 30 fps, 30 tbr, 15360 tbn, 60 tbc (default)
    Metadata:
      handler_name    : VideoHandler
 16.53 M-V: -0.018 fd= 254 aq=    0KB vq=    0KB sq=    0B f=0/0
```

Figura 15: Captura após ffplay

Com a imagem acima observamos que a largura de banda medida pelo *ffplay* foi 878k bits/s, isso corresponde à quantidade de dados comprimidos que estão a ser descodificados e exibidos, sem considerar sobrecargas de rede ou retransmissões.

A largura de banda medida no Wireshark é maior (1057k bits/s) pois inclui o overhead do protocolo devido ao Wireshark capturar o tráfego bruto de rede, que inclui *overheads* adicionais de TCP, IP, Ethernet, retransmissões e controlo de fluxo porque se houver perda de pacotes ou retransmissões (conforme mostrado em imagens anteriores), maior largura de banda será usada para garantir a entrega dos *packets* e levar uma boa experiência ao utilizador.

Questão 4

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 e justifique a largura de banda necessária para que o stream de vídeo sofra alterações.

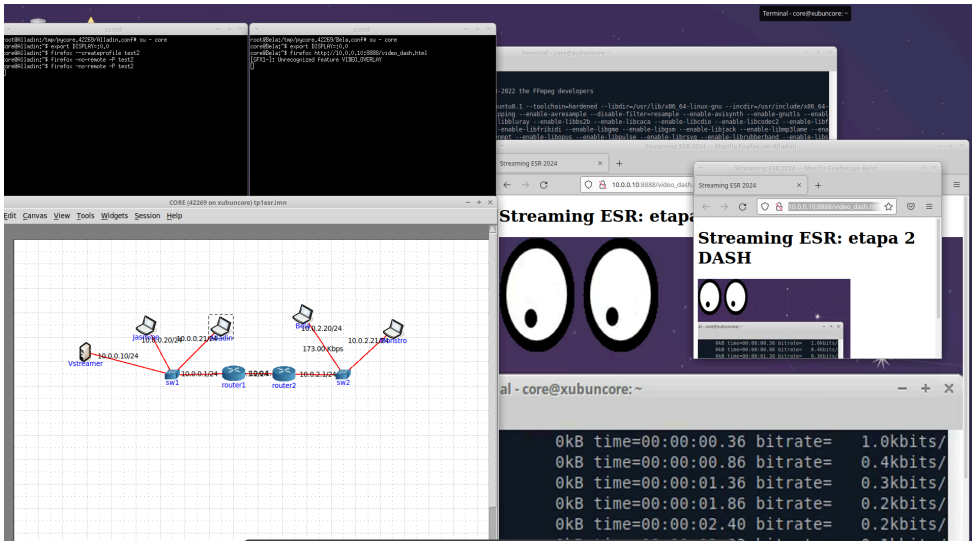


Figura 16: Visualização do dois Firefoxs a correrem em paralelo, um com maior resolução e outro menor, Alladin e Bela respetivamente

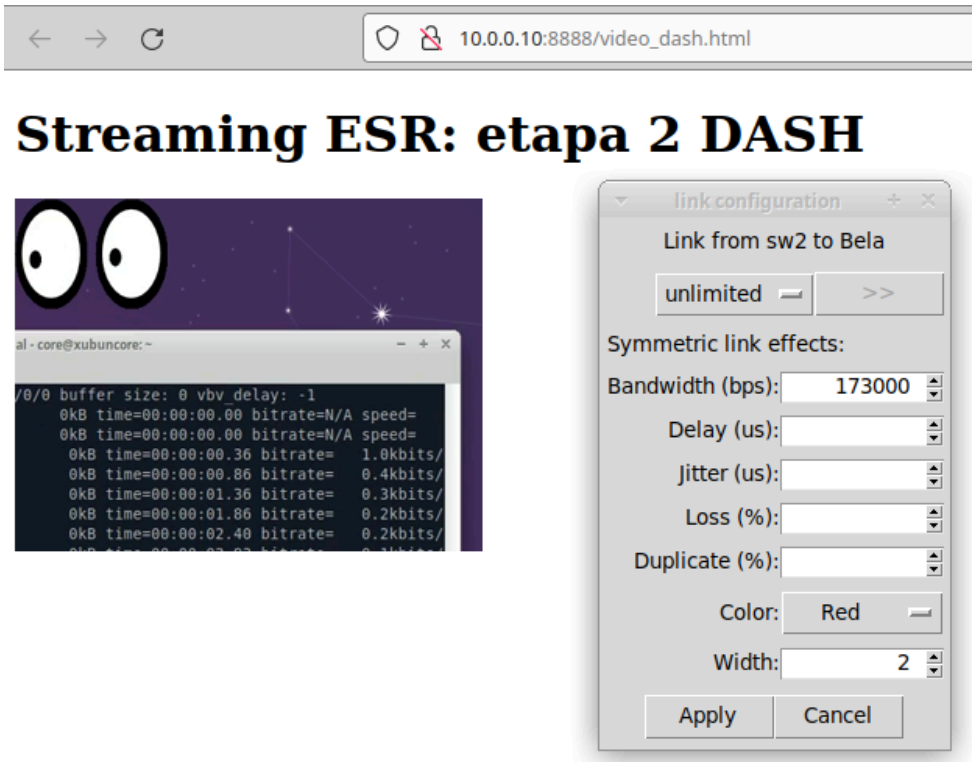
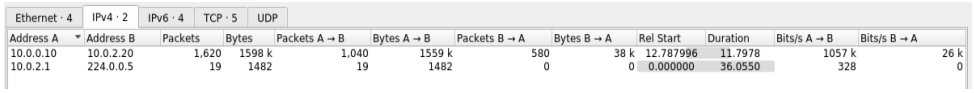


Figura 17: Débito do link entre *Switch 2* e o Cliente com o vídeo de menor resolução(Bela)

Realçar que o valor do link de ligação entre o cliente de melhor resolução (Alladin) foi o mesmo usado na **Questão 2** para limitar a largura de banda para obter o vídeo de maior resolução, ou seja, 1500k bits/s!

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	10.0.2.20	450	363 k	251	348 k	199	14 k	2.979341	17.1211	162 k	
10.0.2.1	224.0.0.5	17	1326	17	1326	0	0	0.000000	32.0175	331	

Figura 18: Captura de Statistics -> Conversations -> IPv4 do video com menor resolução



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	10.0.2.20	1,620	1598 k	1,040	1559 k	580	38 k	12.787996	11.7978	1057 k	26 k
10.0.2.1	224.0.0.5	19	1482	19	1482	0	0	0.000000	36.0550	328	0

Figura 19: Captura de Statistics -> Conversations -> IPv4 do video com maior resolução

A largura de banda necessária para que o stream de vídeo sofra alterações está diretamente relacionada à qualidade de vídeo disponível no servidor e à capacidade do link.

- Bela com uma largura de banda de link reduzida vai receber o vídeo de menor resolução, pois o DASH irá selecionar automaticamente uma qualidade compatível com essa largura de banda.
- Alladin, com uma largura de banda de link maior (a partir de 1.5M bits/s) receberá o vídeo de maior qualidade.

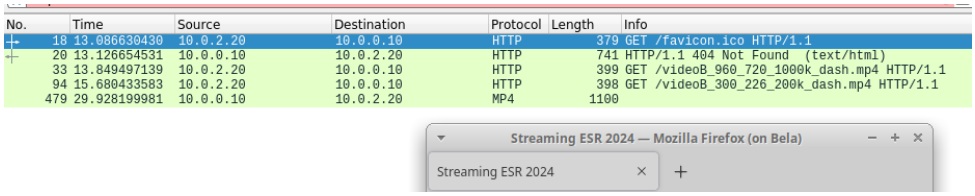


Figura 20: Captura HTTP do vídeo de menor resolução

Como podemos observar na captura acima houve uma tentativa de um *Get* do vídeo de resolução intermédio mas o Dash leva o cliente a tomar a decisão, devido à largura de banda, que é melhor optar pelo vídeo de menor resolução.

Podemos então concluir que ao limitar a conexão a 173k bits/s, um valor ligeiramente superior à largura de banda necessária para *stream* do vídeo de menor resolução, o Dash força o cliente a receber apenas o vídeo de menor resolução.

Ao contrário, o cliente Alladin tendo um limite na largura de banda do link de conexão ligeiramente maior à largura de banda necessária para transmitir o vídeo de maior resolução, o mesmo consegue receber o esse mesmo vídeo sem qualquer restrição!

Uma última observação feita foi que ao limitar as conexões para as exatas larguras de bandas descritas no *video_manifest.mpd*, no caso do cliente Bela com o vídeo de menor resolução, o mesmo não conseguia reproduzir nenhum dos vídeos e no caso do cliente Alladin ele reproduzia o vídeo de resolução intermédia, concluindo que devido a retransmissões e perdas de pacotes é necessário haver algum intervalo entre a largura de banda mínima descrita no *manifest* e o valor de facto usado no link de conexão.

Questão 5
Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado e comparando o modelo de streaming com o que foi utilizado na Questão 1.

Quando o cliente tenta aceder à página recebe do servidor o ficheiro MPD que possui a informação relativa à largura de banda mínima requerida pelos três diferentes vídeos, tendo em conta a largura de banda de ligação entre servidor e cliente escolhe a resolução que consegue suportar, levando a que ao diminuir a largura de banda da ligação a resolução do vídeo diminuía também.

Na questão 1 foi usado o modelo de streaming HTTP simples, o que leva a que o vídeo seja transmitido numa resolução fixa, diminuindo a largura de banda ou face a problemas de ligação o cliente poderá experienciar uma má qualidade de visualização da *stream* causando uma má experiência para o cliente.

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

Questão 6

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 também para os cenários de 1000 e 10000 clientes.

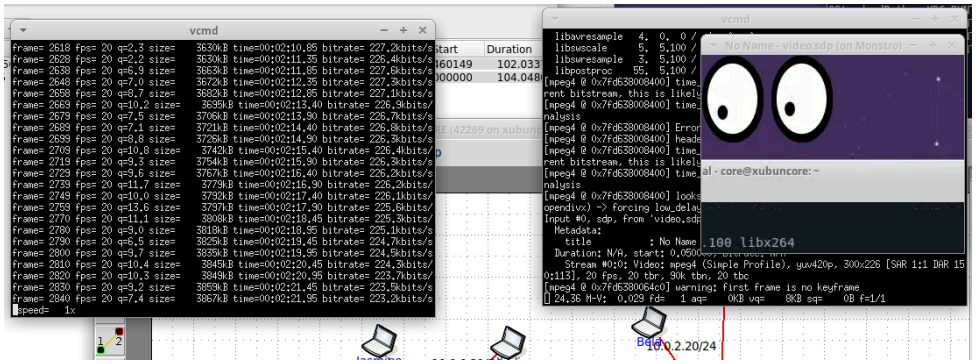


Figura 21: Streaming *Unicast*

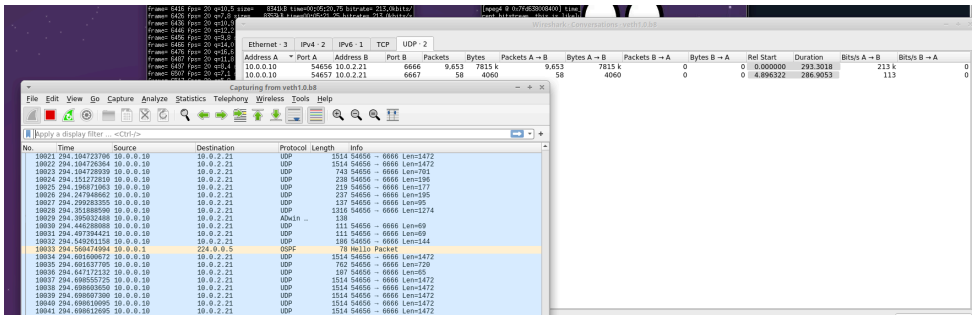


Figura 22: Captura do tráfego *Unicast*

Após criar uma sessão de *streaming* RTP com *ffmpeg* e *ffplay* no cliente "Monstro", capturamos o tráfego com o *Wireshark* no link de saída do servidor, podemos observar nas capturas acima que o débito de *Unicast* é de 213k bits/s

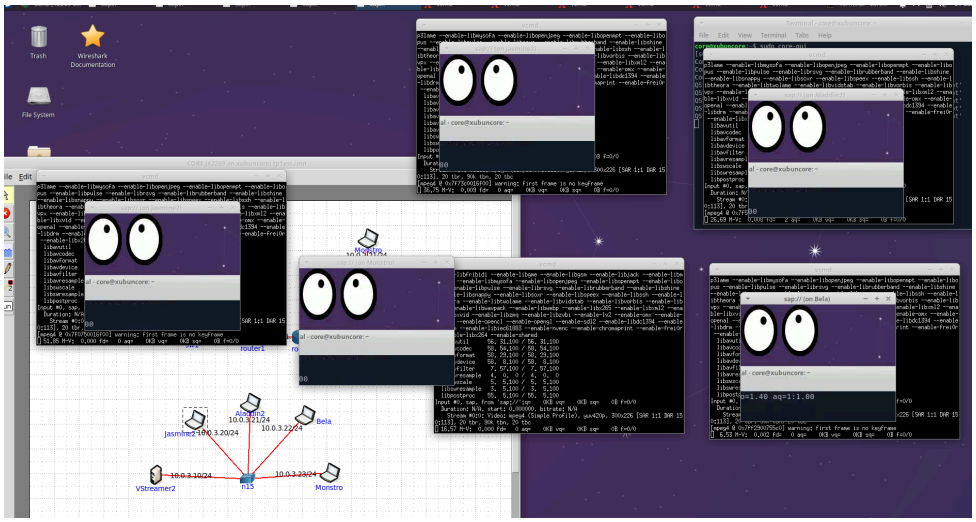


Figura 23: Streaming *Multicast*

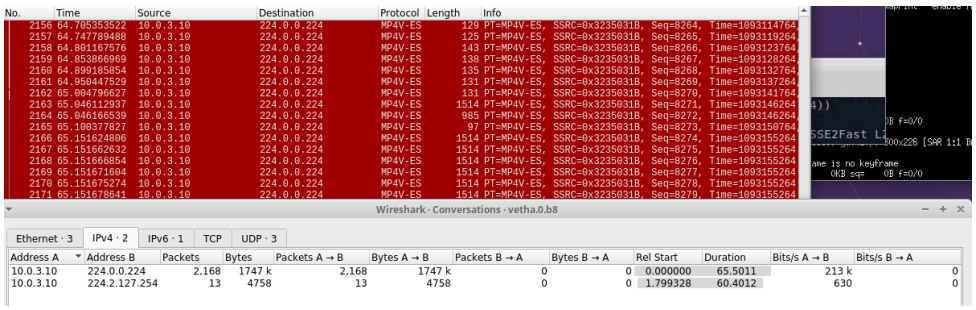


Figura 24: Captura de tráfego *Multicast*

Após ser criada a nova topologia, iniciámos com recurso a *ffmpeg* uma sessão de streaming *multicast* para cada cliente e capturamos o tráfego no link de saída do servidor do Wireshark.

Observamos que o débito foi de, igualmente, 213k bits/s.

Com isto, podemos concluir que o *multicast* com um igual débito consegue satisfazer maior número de clientes, sendo superior em termos de escabilidade ao *unicast*. Para além disso, em *unicast*, para calcular o débito total multiplica-se o débito de um cliente pelo número total de clientes, ao contrário em *multicast* o servidor envia uma única cópia do fluxo de dados, que é distribuída por toda a rede sendo mais eficiente aumentar o número de clientes em *mulicast*. Em suma, em caso de *unicast* como cada cliente recebe uma cópia exclusiva do fluxo de vídeo diretamente do servidor, para N clientes existirão N fluxos, tendo uma escabilidade do débito linear enquanto o *multicast* mantém-se constante.

Para um cenário de streaming para 1000 e até mesmo 10000 clientes podemos concluir que *multicast* seria a opção mais viável, apesar de ser importante realçar que *multicast* exige uma infraestrutura necessária, a rede precisa ser configurada para suportar *multicast*, sendo mais custoso e díficil de implementar e menos flexibilidade pois todos os clientes que fazem parte do grupo *multicast* recebem o mesmo conteúdo e qualidade.

Conclusões

Concluindo este trabalho prático, onde utilizamos diferentes métodos de *stream* de vídeo, como HTTP simples, MPEG-DASH e Multicast, podemos tirar diferentes conclusões sobre escalabilidade, eficiência e qualidade de serviço em diferentes contextos.

As mais importantes foram sem dúvida como HTTP simples para cenários com elevados clientes acaba por não ser ideal pois o seu débito e tráfego escala linearmente com o número de clientes.

MPEG-DASH acaba por ser uma boa solução principalmente para lidar com diferentes condições de rede, levando a que os utilizadores mesmo com diferentes problemas de rede possam ter uma experiência fluida e satisfatória, entregando vídeos de menor resolução dependo da largura de banda da ligação com o cliente graças ao ficheiro MPD, ao contrário do HTTP simples que face a menores larguras de banda irá enviar sempre o mesmo vídeo constantemente podem causar problemas de *buffering* como travagens no vídeo.

Quanto às abordagens *unicast* e *multicast*, ambas trouxeram as suas vantagens, como por exemplo *unicast* que é eficiente para pequenos números de clientes, acaba por não ser viável à mediada que o número de clientes cresce, pois à medida que o número de clientes cresce o número de fluxos cresce igualmente, aumentado proporcionalmente o débito. O que é compensado pela metodologia *multicast* que acaba por ser uma melhor solução em termos de escalabilidade mantendo um débito constante independentemente do número de clientes.

Resumindo, as experiências realizadas destacaram a importância de escolher a tecnologia de *streaming* adequada ao contexto e às necessidades pretendidas. Soluções como o MPEG-DASH ou *multicast* são mais viáveis para ambientes de maior escala, garantindo tanto a eficiência da rede quanto a qualidade do serviço apesar de terem por norma implementações mais custosas, ao contrário métodos como *unicast* e HTTP simples acabam por ter implementações mais simples e são satisfatórias quando há menores números de clientes e garantias de estabilidade de conexão de rede.