



Universidade do Minho

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Engenharia de Serviços em Rede

Ano Letivo de 2021/2022

Streaming de áudio e vídeo a pedido e em tempo real

João Amorim, A74806

Tiago Matos, PG45585

Gonçalo Costeira, PG47219

30 de dezembro de 2021

ERS

Índice

1 Streaming HTTP simples sem adaptação dinâmica de débito

- 1.1 Tarefas
- 1.2 Questões
 - 1.2.1 1. Capture três pequenas amostras de tráfego no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (VLC e Firefox) e com 3 clientes (VLC, Firefox e ffmpeg). Identifique a taxa em bps necessária (usando o ffmpeg -i video1.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)

2 Streaming adaptativo sobre HTTP (MPEG-DASH)

- 2.1 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.
- 2.2 Ajuste o débito dos links da topologia de modo que o cliente no portátil 2 exiba o vídeo de menor resolução e o cliente no portátil 1 exiba o vídeo com mais resolução. Mostre evidências.
- 2.3 Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado.

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

- 3.1 Tarefas
- 3.2 Questões
 - 3.2.1 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.

4 Conclusões

Anexos

1 Streaming HTTP simples sem adaptação dinâmica de débito

1.1 Tarefas

Nesta primeira Etapa, pretende-se testar o streaming sobre HTTP sem adaptação dinâmica de débito. Inicialmente, vamos começar por criar a nossa topologia, de acordo com o enunciado. A topologia irá ter um servidor, 3 portáteis, 2 switches e dois routers.

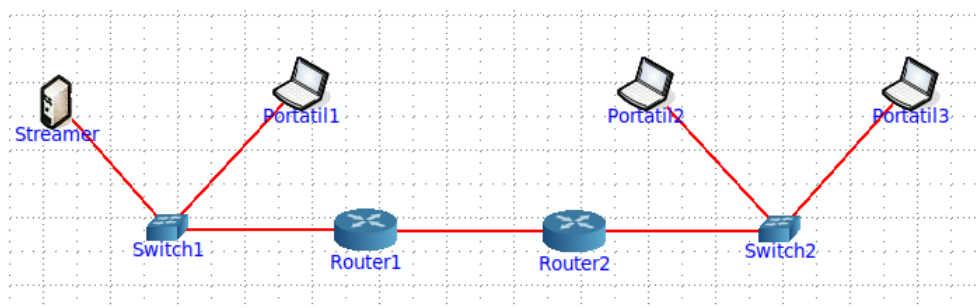


Figura 1.1: Topologia criada no CORE

De seguida, e para garantir que a topologia criada funciona sem problemas, vamos testar a conectividade da mesma, com o nmap:

```
root@Streamer:/tmp/pycore.39983/Streamer.conf# nmap -n -sP 10.0.*/*24
Starting Nmap 7.80 ( https://nmap.org ) at 2021-11-08 00:34 WET
Nmap scan report for 10.0.0.1
Host is up (0.000038s latency).
MAC Address: 00:00:00:AA:00:02 (Xerox)
Nmap scan report for 10.0.0.20
Host is up (0.000080s latency).
MAC Address: 00:00:00:AA:00:01 (Xerox)
Nmap scan report for 10.0.0.10
Host is up.
Nmap scan report for 10.0.1.1
Host is up (0.000014s latency).
Nmap scan report for 10.0.1.2
Host is up (0.000030s latency).
Nmap scan report for 10.0.2.1
Host is up (0.000029s latency).
Nmap scan report for 10.0.2.20
Host is up (0.00024s latency).
Nmap scan report for 10.0.2.21
Host is up (0.00037s latency).
```

Figura 1.2: Comando nmap -n -sP

Como podemos ver, a topologia encontra-se a funcionar tal como pretendido.

Nos próximos passos, nomeadamente do 3 ao 9, é feita uma explicação sobre como iniciar e efetuar o streaming para os vários clientes. No caso desta etapa, irá ser feito o streaming do video1.mp4, a partir de um servidor a funcionar com VLC e com o nome Streamer. Quanto aos clientes, estes têm o nome de Portatil1, Portatil2 e Portatil 3, em que o primeiro corresponde a um outro VLC, o segundo a uma pagina no Firefox e o terceiro a uma instância do ffmpeg.

Ao longo da execução, serão então efetuadas 3 capturas no link de saída do servidor para os seguintes clientes:

- 1ª Captura - Portatil1 (VLC)
- 2ª Captura - Portatil1 (VLC) + Portatil2 (Firefox)
- 3ª Captura - Portatil1 (VLC) + Portatil2 (Firefox) + Portatil3 (ffmpeg)

O resultado obtido foi o seguinte:



Figura 1.3: Resultado da execução das tarefas. Cliente VLC + Client Firefox + Cliente ffplay

1.2 Questões

1.2.1 1. Capture três pequenas amostras de tráfego no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (VLC e Firefox) e com 3 clientes (VLC, Firefox e ffmpeg). Identifique a taxa em bps necessária (usando o ffmpeg -i video1.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)

Em primeiro lugar, vamos utilizar o comando ffmpeg -i video1.mp4 para verificar a taxa em bps que é de facto necessária:

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'video1.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  encoder          : Lavf58.29.100
Duration: 00:00:15.75, start: 0.000000, bitrate: 17 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 180x120, 14 kb/s, 20
fps, 20 tbr, 10240 tbn, 40 tbc (default)
Metadata:
  handler name     : VideoHandler
```

Figura 1.4: Comando ffmpeg -i video.mp4

Tal como se vê na imagem, a taxa é de 17kb por segundo.

No que diz respeito ao encapsulamento e ao número de fluxos, recorreremos às ferramentas do Wireshark. Para isto, utilizamos a "Protocol Hierarchy" em "Statistics", para identificarmos o tipo de encapsulamento usado, enquanto que para o número de fluxos, utilizamos a ferramenta "Conversations", também esta em "Statistics". Os resultados obtidos foram os seguintes:

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
▼ Frame	100.0	1661	100.0	1134380	116 k	0	0
▼ Ethernet	100.0	1661	2.0	23254	2395	0	0
▼ Internet Protocol Version 6	0.5	8	0.0	320	32	0	0
Open Shortest Path First	0.5	8	0.0	288	29	8	288
▼ Internet Protocol Version 4	99.3	1649	2.9	32980	3397	0	0
▼ Transmission Control Protocol	96.9	1610	94.8	1075710	110 k	1455	855714
▼ Hypertext Transfer Protocol	9.3	155	19.2	217802	22 k	140	205600
Malformed Packet	0.9	15	0.0	0	0	15	0
Open Shortest Path First	2.3	39	0.2	1716	176	39	1716
Address Resolution Protocol	0.2	4	0.0	112	11	4	112

Figura 1.5: Protocol Hierarchy - Encapsulamento utilizado

Wireshark · Conversations · Captura3.pcapng											
Ethernet · 4		IPv4 · 4		IPv6 · 1		TCP · 3		UDP			
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	
10.0.0.20	48064	10.0.0.10	8080	628	439 k	314	20 k	314	418 k	0.00000	
10.0.2.20	51492	10.0.0.10	8080	628	439 k	314	20 k	314	418 k	0.00029	
10.0.2.21	34218	10.0.0.10	8080	628	439 k	314	20 k	314	418 k	0.00010	

Figura 1.6: Conversations - Número de Fluxos para a Captura com os 3 Clientes em funcionamento.

Para efeito de capturas no Wireshark, todas as três capturas tiveram relativamente a mesma duração, cerca de 30 segundos, correspondendo à duração de Streaming do video1.mp4 por duas vezes. No entanto, e de acordo com as capturas efetuadas, existem atrasos no streaming do video1, para a segunda captura em relação à primeira e para a terceira captura em relação à segunda. Isto indica-nos que este atraso se iria acentuar caso fossem adicionados mais Clientes, fazendo-nos concluir que esta solução terá realmente problemas de escalabilidade.

2 Streaming adaptativo sobre HTTP (MPEG-DASH)

2.1 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.

Conforme podemos verificar nos parâmetros do ficheiro MPD, a largura de banda mínima necessária para a visualização do streaming nas várias resoluções é a seguinte

- Low resolution: 148kbps;
- Medium resolution: 382kbps;
- High resolution: 734kbps;

```
<Title>video_manifest.mpd generated by GPAC</Title>
</ProgramInformation>
- <Period duration="PT0H0M12.734S">
- <AdaptationSet segmentAlignment="true" bitstreamSwitching="true"
maxWidth="540" maxHeight="360" maxFrameRate="30" par="3:2" lang="und">
- <SegmentList>
<Initialization sourceURL="video_manifest_init.mp4"/>
</SegmentList>
+ <Representation id="1" mimeType="video/mp4" codecs="avc3.64000c"
width="180" height="120" frameRate="30" sar="1:1" startWithSAP="0"
bandwidth="147770">
</Representation>
+ <Representation id="2" mimeType="video/mp4" codecs="avc3.640014"
width="360" height="240" frameRate="30" sar="1:1" startWithSAP="0"
bandwidth="381882">
</Representation>
+ <Representation id="3" mimeType="video/mp4" codecs="avc3.64001e"
width="540" height="360" frameRate="30" sar="1:1" startWithSAP="0"
bandwidth="734288">
</Representation>
</AdaptationSet>
```

Figura 2.1: MPD file

Em termos de pilha protocolar é utilizado HTTP, um dos protocolos mais utilizados em conteúdo media, sobre TCP, um protocolo de controlo de fluxo, importante para garantir a comunicação fiável e estável, neste caso entre servidor de streaming e os clients.

2.2 Ajuste o débito dos links da topologia de modo que o cliente no portátil 2 exiba o vídeo de menor resolução e o cliente no portátil 1 exiba o vídeo com mais resolução. Mostre evidências.

Seguindo as resoluções mínimas verificadas no exercício anterior, de forma a que o PC2 reproduza o video de menor resolução é necessário uma limitação do link entre os 148-382kbps, limitamos o link entre o PC2 e switch a 256kbps. No outro caso, PC1 reproduzir o video com maior resolução, o link tem que ter no mínimo uma largura de banda de 734kbps, configuramos uma largura de banda de 1100kbps no link entre o PC1 e o switch. Esta limitação poderia ser feita entre Routers e/ou switches, mas iria causar um bottleneck afetando os vários PCs/clients da rede.

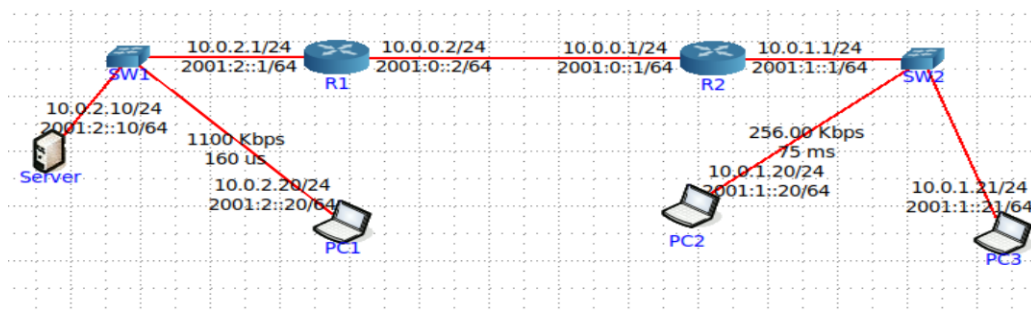


Figura 2.2: Topologia da Rede

Como podemos visualizar na captura de wireshark, no PC1 confirmamos que o video a ser reproduzido é o de menor resolução e no PC2 o de maior resolução.

A captura de pacotes no Wireshark mostra as seguintes requisições de vídeo:

No.	Time	Source	Destination	Protocol	Length	Info
14	5.251793985	10.0.2.20	10.0.2.10	HTTP	498	GET /favicon.ico HTTP/1.1
16	5.261178432	10.0.2.10	10.0.2.20	HTTP	741	HTTP/1.1 404 Not Found (text)
23	5.286321561	10.0.2.20	10.0.2.10	HTTP	447	GET /video_manifest.mpd HTTP/1.1
34	6.201718866	10.0.2.20	10.0.2.10	HTTP	390	GET /video_manifest_init.mp4
36	6.211927743	10.0.2.10	10.0.2.20	MP4	1060	
43	6.560582665	10.0.2.20	10.0.2.10	HTTP	420	GET /video2_540_360_1000k_das
1442	15.722941442	10.0.2.20	10.0.2.10	HTTP	422	GET /video2_540_360_1000k_das
2842	24.845451996	10.0.2.20	10.0.2.10	HTTP	422	GET /video2_540_360_1000k_das

Figura 2.3: Streaming video PC1

No.	Time	Source	Destination	Protocol	Length	Info
57	73.946260491	10.0.1.20	10.0.2.10	HTTP	400	GET /favicon.ico HTTP/1.1
62	74.060422956	10.0.2.10	10.0.1.20	HTTP	741	HTTP/1.1 404 Not Found (text)
74	75.599453772	10.0.1.20	10.0.2.10	HTTP	420	GET /video2_540_360_1000k_dash
1507	117.781252711	10.0.1.20	10.0.2.10	HTTP	420	GET /video2_180_120_200k_dash
1798	126.755564365	10.0.1.20	10.0.2.10	HTTP	421	GET /video2_180_120_200k_dash

Figura 2.4: Streaming video PC2

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

Neste exercício verificamos o correto funcionamento do DASH (Dynamic Adaptive Streaming over HTTP), onde a resolução do streaming no cliente adapta-se consoante a largura de banda disponível até ao servidor. O servidor de streaming tem disponível várias versões com diferentes resoluções ou bitrates, que são descritas no manifest file MPD. O MPD file é downloaded pelo cliente, na primeira tentativa o servidor de streaming envia o vídeo com melhor resolução, caso a largura de banda permita, como no caso do PC1, ele continua a enviar o streaming com maior resolução, caso a largura de banda não permita, exemplo PC2, através do DASH existe a adaptação para o vídeo com a melhor resolução possível conforme descrito no manifest file.

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

3.1 Tarefas

Nesta última etapa o streaming será sempre feito sobre UDP, quer em unicast, quer em multicast. No primeiro caso, o cliente recebe dados sobre a sessão de streaming num ficheiro de descrição em formato SDP (Session Description Protocol). O servidor ao iniciar a stream, gera o ficheiro, que deveria ser depois fornecido ao cliente, mas que neste caso não é necessário, pois o sistema de ficheiros é partilhado. O cliente com base nos dados contidos no ficheiro, inicia a receção em UDP.

Já no caso multicast o modo de funcionamento é distinto. Também é gerada uma descrição da sessão, mas ela é enviada por SAP (Session Announcement Protocol) para um grupo multicast (endereço de grupo) especial para o efeito que é o 224.2.127.254 (sap.mcast.net) para destinos IPv4 ou ff0e::2:7ffe para destinos IPv6. Tudo o que o cliente tem de fazer é estar à escuta nesse grupo e ouvir o anúncio com os dados SDP para poder iniciar a sessão como cliente. No caso multicast, vamos escolher também um endereço de envio especial, que é um endereço de grupo. Neste exemplo foi escolhido o 224.0.0.200 porta 5555.

Para responder às questões foram realizadas uma série de tarefas para recolher a informação necessária. Para o streaming unicast podemos ver nas figuras 4.1, 4.2 e 4.3 em anexo. Já para o streaming multicast podemos ver nas figuras 4.4 e 4.5 em anexo.

3.2 Questões

3.2.1 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.

A transmissão multicast é um método de transmissão 1 para N entre um servidor e N clientes. O servidor multicast envia uma única stream, permitindo que vários clientes captem. Não há relação direta entre os clientes e o servidor. Os clientes subscrevem um grupo multicast e a rede garante a entrega da stream. Desta forma, não há sobrecarga adicional no servidor multicast se um novo cliente subscrever o grupo multicast. O servidor terá a mesma carga quer esteja 1 ou 10 000 clientes.

Unicast é uma conexão end to end. Como o servidor unicast tem uma ligação direta com cada cliente, cada novo cliente irá ocupar mais largura de banda. Devido ao aumento do consumo de rede, o unicast não é adequado para aplicações em que vários clientes estão a receber o mesmo conteúdo simultaneamente.

Como podemos ver nas figuras 4.6 e 4.7 praticamente não existe redução na velocidade de transmissão. E são enviados exatamente os mesmos packages por segundo como seria de esperar. Então podemos concluir que a nível de escalabilidade uma solução multicast seria o mais apropriado.

4 Conclusões

Com este trabalho conseguimos então perceber as várias soluções de streaming solicitado ou em real-time, as diferenças conceptuais entre as várias opções disponíveis em termos de pilhas protocolares sendo elas desde HTTP sobre TCP e ou RTP sobre UDP. Aprendemos também a utilizar várias ferramentas open-source tais como, CORE Emulator, Wireshark, FFmpeg, VideoLAN VLC e OBS Studio.

No primeiro ponto, onde era pretendido testar o streaming sobre HTTP sem DASH, verificamos que há medida que aumentamos o número de clientes a visualizar o streaming, o delay dos streamings nos vários clientes vai aumentando, verificando assim que a solução não é escalável.

Relativamente ao segundo ponto sobre o DASH, verificamos como é o seu funcionamento, como é efetuada a criação do conteúdo em diferentes resoluções/bitrates no servidor de streaming, assim como a criação do ficheiro MPD onde podemos verificar a descrição de cada vídeo de modo que o cliente saiba quais as resoluções existentes. Verificamos o comportamento nos vários clientes com a adaptação dinâmica do streaming dependendo da largura de banda disponível entre os PCs e o servidor.

No último ponto, Streaming unicast RTP/RTCP e streaming multicast, ambos sobre UDP, verificamos que no caso de unicast como é uma conexão end-to-end, o consumo de largura de banda irá ser tanto maior quando maior for o número de clientes. No caso do multicast, como é uma ligação do servidor para N clientes, o servidor terá a mesma carga. A nível de escalabilidade, obviamente a solução unicast é a mais apropriada.

Referências

[Kurose and Ross, 2021] Kurose, J. F. and Ross, K. W. (2021). Computer networking, a top-down approach 7th.

Anexos

```
vcmd
core85Stream:~$ ^C
core85Stream:~$ ffmpeg -stream_loop -1 -re -i video1.mp4 -f rtp -sdp_file video.sdp rtp://10.0.2.21:5555
ffmpeg version 4.2.4-lubuntu0.1 Copyright (c) 2000-2020 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.3.0-10ubuntu2)
  configuration: --prefix=/usr --extra-version=lubuntu0.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu
  --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-avresample --disab
  le-filter=resample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-lib
  bluray --enable-libbs2b --enable-libbaca --enable-libcdio --enable-libcodec2 --enable-libflite --enable-libfontco
  nfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame
  --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-libsvg
  --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --
  enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwavpack
  --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-libzvi --enable-lv2
  --enable-lv2 --enable-opengl --enable-opengl --enable-opengl --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm --
  enable-libiec61883 --enable-nvenc --enable-nvenc --enable-chromaprint --enable-frei0r --enable-libx264 --enable-shared
  libavutil      58, 31.100 / 58, 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   56,  5.100 / 56,  5.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'video1.mp4':
  Metadata:
    major_brand      : isom
    minor_version    : 512
    compatible_brands: isoiso2avc1mp41
    encoder          : Lavf58.29.100
  Duration: 00:00:19.05, start: 0.000000, bitrate: 15 kb/s
  Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637651), yuv420p, 180x120, 13 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
  Metadata:
    handler_name     : VideoHandler
Stream mapping:
  Stream #0:0 -> #0:0 (h264 (native) -> mpeg4 (native))
Press [q] to stop, [?] for help
Output #0, rtp, to 'rtp://10.0.2.21:5555':
  Metadata:
    major_brand      : isom
    minor_version    : 512
    compatible_brands: isoiso2avc1mp41
    encoder          : Lavf58.29.100
  Stream #0:0(und): Video: mpeg4, yuv420p, 180x120, q=2-31, 200 kb/s, 20 fps, 90k tbn, 20 tbc (default)
  Metadata:
    handler_name     : VideoHandler
    encoder          : Lavc58.54.100 mpeg4
  Side data:
    cpb: bitrate max/min/avg: 0/0/2000000 buffer size: 0 vbv_delay: -1
frame= 1132 fps= 20 q=2.0 size=    979kB time=00:00:56.95 bitrate= 141.8kbits/s speed=0.997x
```

Figura 4.1: Inicio da sessão de streaming unicast com RTP no streamer

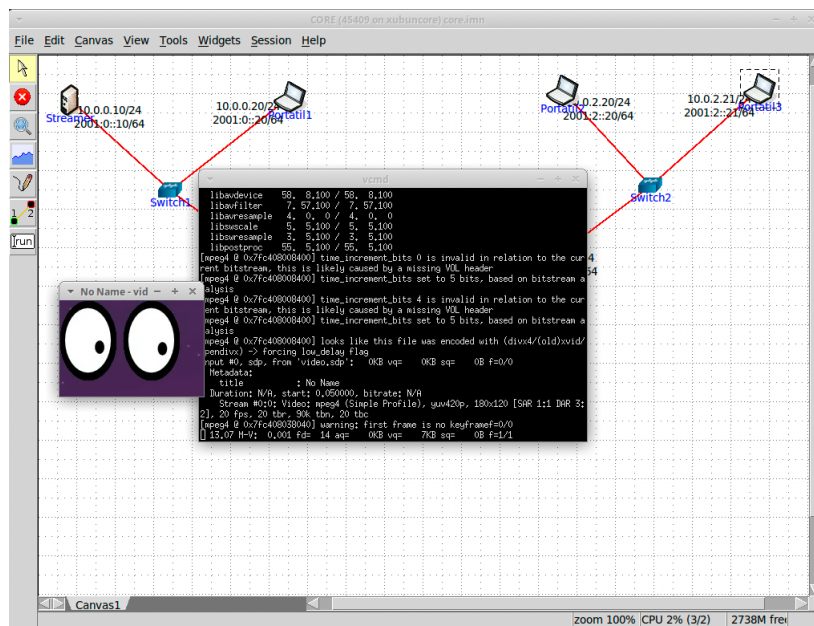


Figura 4.2: ffplay no cliente Portátil 3

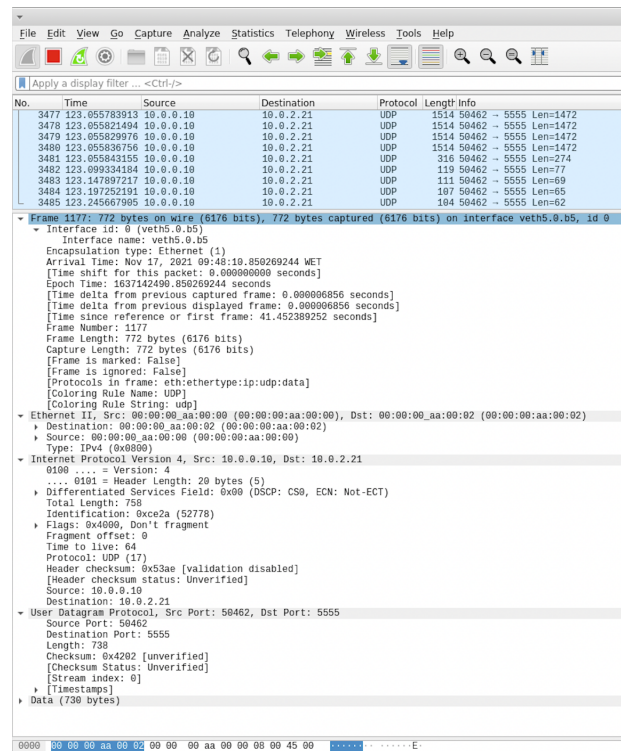


Figura 4.3: Captação do tráfego no wireshark em unicast

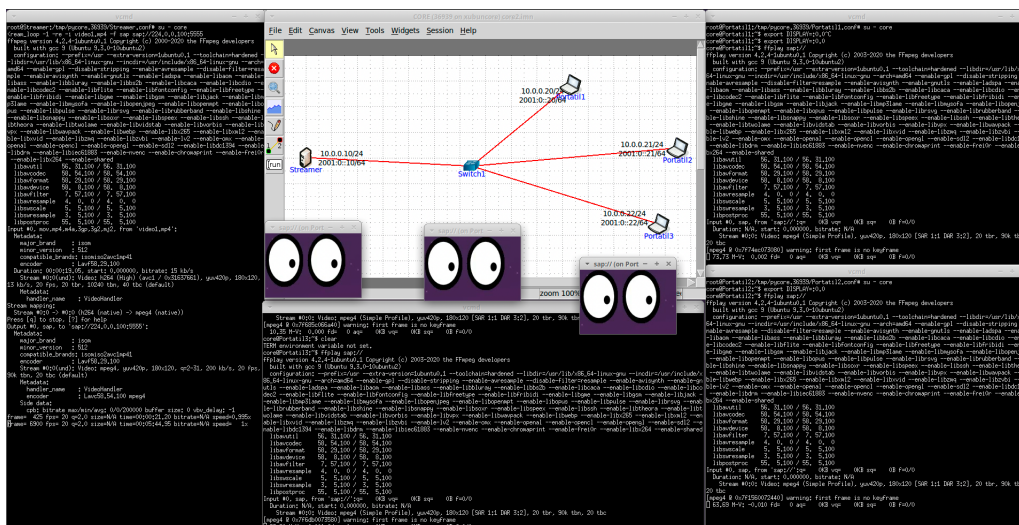


Figura 4.4: Topologia multicast com o streaming a funcionar

No.	Time	Source	Destination	Protocol	Length	Info
1819	65.29725868	19.0.0.10	224.0.0.100	MP4V-ES	116	Pf=MP4V-ES, SSRC=0xCDAFB17, Seq=15681, Time=135031173, Mark
1820	65.34925486	19.0.0.10	224.0.0.100	MP4V-ES	106	Pf=MP4V-ES, SSRC=0xCDAFB17, Seq=15682, Time=1350316273, Mark
Frame 1: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface veth2.0.0b, id 0 Interface 10: 0 (veth2.0.0b) Encapsulation type: Ethernet (1) Arrival Time: Nov 17, 2021 10:03:21.896293392 MET [Time shift for this packet: 0.000000000 seconds] Epoch Time: 1637434401.896293392 seconds [Time delta from previous captured frame: 0.000000000 seconds] [Time delta from previous displayed frame: 0.000000000 seconds] [Time since reference or first frame: 0.000000000 seconds] Frame Number: 1 Frame Length: 104 bytes (832 bits) Capture Length: 104 bytes (832 bits) [Frame is marked: False] [Frame is ignored: False] [Protocols in frame: eth:ethertype:ip:udp:data] [Coloring Rule Name: TTL low or unexpected] [Coloring Rule String: (1 ip.dst == 224.0.0.0/4 && ip.ttl < 5 && ip.m < 5 && !ospf) (ip.dst == 224.0.0.0/24 && ip.dst != 224.0.0.251 && ; Ethernet II, Src: 00:00:00:aa:00:00 (00:00:00:aa:00:00), Dst: IPv4cast.64 (01:00:5e:00:00:64) Destination: IPv4cast.64 (01:00:5e:00:00:64) Source: 00:00:00:aa:00:00 (00:00:00:aa:00:00) Type: IPv4 (0x0009) Internet Protocol Version 4, Src: 19.0.0.10, Dst: 224.0.0.100 0100 = Version: 4 ... 0101 Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 90 Identification: 0x00b0 (39856) Flags: 0x0000, Don't fragment Fragment offset: 0 Time to live: 255 Protocol: UDP (17) Header checksum: 0xf573 [validation disabled] [Header checksum status: Unverified] Source: 19.0.0.10 Destination: 224.0.0.100 User Datagram Protocol, Src Port: 58364, Dst Port: 5555 Source Port: 58364 Destination Port: 5555 Length: 70 Checksum: 0xebd7 [unverified] [Checksum Status: Unverified] [Stream index: 0] [Timestamps] Data (62 bytes)						

Figura 4.5: Captação do tráfego no wireshark em multicast

Interfaces				
Interface	Dropped packets	Capture filter	Link type	Packet size limit
veth5.0.b5	Unknown	none	Ethernet	262144 bytes
Statistics				
Measurement	Captured	Displayed	Marked	
Packets	6495	6495 (100.0%)	—	
Time span, s	229.696	229.696	—	
Average pps	28.3	28.3	—	
Average packet size, B	664	664	—	
Bytes	4309772	4309772 (100.0%)	0	
Average bytes/s	18 k	18 k	—	
Average bits/s	150 k	150 k	—	

Figura 4.6: Estatísticas wireshark em unicast

Interfaces				
<u>Interface</u>	<u>Dropped packets</u>	<u>Capture filter</u>	<u>Link type</u>	<u>Packet size limit</u>
veth2.0.db	Unknown	none	Ethernet	262144 bytes
Statistics				
<u>Measurement</u>	<u>Captured</u>	<u>Displayed</u>	<u>Marked</u>	
Packets	3235	3235 (100.0%)	—	
Time span, s	116.150	116.150	—	
Average pps	27.9	27.9	—	
Average packet size, B	668	668	—	
Bytes	2159621	2159621 (100.0%)	0	
Average bytes/s	18 k	18 k	—	
Average bits/s	148 k	148 k	—	

Figura 4.7: Estatísticas wireshark em multicast