Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Adaptive vídeo streaming, QoS and traffic control

Serviços de Comunicações

2019/2020

Hugo Miguel Malheiro de Passos Guia - up201305075

João Carlos Couto Antunes Fonseca - up201403802

Índice

1) Análise dos ficheiros MPD.…… **2**

1.1) Primeiro ficheiro.……. **2**

1.2) Segundo ficheiro.……. **3**

1.3) Terceiro ficheiro.…….. **4**

2) Análise via *Wireshark*.…… **5**

2.1) *Web Player* (*bitmovin*)……. **5**

2.2) Cliente local (*Shaka Player*).……... **6**

3) Implementação de uma estrutura de rede para reprodução de vídeo com interferência via iperf… **6**

3.1) Topologia.…… **6**

3.2) *Iperf*.……. **7**

3.2.1) Primeira tentativa: UDP.……... **8**

3.2.2) Segunda tentativa: TCP.…… **10**

4) Controlo de tráfego via TC.……. **10**

4.1) Aplicação.……. **13**

5) Conclusão.……... **15**

1) Análise dos ficheiros MPD

1.1) Primeiro ficheiro

O primeiro ficheiro tem como nome manifestARD.mpd. É utilizado o *codec* mp4a.40.2 para o áudio e para o vídeo há uma pletora de *codecs* dispostos na tabela seguinte.

Id	Codec	Bandwidth (bits/s)	Player dimensions (px)	Frame Rate	Audio Sampling Rate
video_00	avc1.640020	2000 000	1280x720	25/1	
video_01	avc1.4d4020	1800 000	960x540	25/1	
video_02	avc1.4d401e	1024 000	640x360	25/1	
video_03	avc1.4d4015	512 000	512x288	25/1	
video_04	avc1.42c015	256 000	480x270	25/1	
video_05	avc1.42c00c	128 000	320x180	25/1	
audio_06	mp4a.40.2	192 000			48000

Tabela 1 - representações disponíveis

Após análise dos dados apresentados, pode-se concluir que o valor de *bandwidth* utilizada pelo áudio (audio_06) é fixa e, como tal, a qualidade do vídeo tem de ser ajustada conforme a *bandwidth* disponível, de forma a não ultrapassar o valor máximo. Como tal,

- 320 Kbits/s, tem de ser usado o id video_05
- 500 Kbits/s, pode-se optar pelo id video_04 (melhor qualidade) ou video_05
- 1.5 Mbits/s, pode-se optar pelo id video_05, video_04 e video_03 (melhor qualidade)
- 10 Mbits/s, qualquer representação é valida, podendo-se optar por video_00 para a melhor qualidade de reprodução

Os *profile files* que se encontram no ficheiro MPD são:

- `urn:hbbtv:dash:profile:isoff-live:2012`, um perfil DASH HbbTV 1.5
- `urn:mpeg:dash:profile:isoff-live:2011`, um identificador para perfil MPEG-DASH ISO *Base media file format live*
- `urn:dvb:dash:profile:dvb-dash:2014`, um perfil DASH HbbTV 2.0

1.2) Segundo ficheiro

O ficheiro que abordamos de seguida, f08e80da-bf1d-4e3d-8899-f0f6155f6efa.mpd, utiliza também o *codec* mp4a.40.2 para o áudio e o *codec* avc1.42c00d para vídeo.

Id	Bandwidth (bits/s)	Player dimensions (px)	Frame Rate	Audio Sampling Rate
180_250000	250 000	320x180	25/1	
270_400000	400 000	480x270	25/1	
360_800000	800 000	640x360	25/1	
540_1200000	1200 000	960x540	25/1	
720_2400000	2400 000	1280x720	25/1	
1080_4800000	4800 000	1920x1080	25/1	
1_stereo_128000	128 000			48000

Tabela 2 - representações disponíveis

Tal como no ficheiro anterior, o valor de *bandwidth* utilizada pelo áudio (1_stereo_128000) é fixa e, também, a qualidade do vídeo terá de ser ajustada conforme a *bandwidth* que se encontra disponível para não ultrapassar o limite permitido. Assim sendo, para um cenário de:

- 320 Kbits/s, não há *bandwidth* suficiente para o áudio e qualquer das representações de vídeo

- 500 Kbits/s, o único com largura de banda suficiente para áudio e vídeo é o id 180_250000
- 1.5 Mbits/s, pode-se optar pelo id 540_1200000 (melhor qualidade), id 360_800000, id 270_400000 e id 180_250000
- 10 Mbits/s, qualquer representação é valida, sendo aquela com o id de 1080_4800000 a que aufere melhor qualidade

Há um *profile file* único neste ficheiro MPD intitulado de `urn:mpeg:dash:profile:isoff-main:2011` que se trata de um perfil principal do tipo *MPEG-DASH ISO Base media file format.*

1.3) Terceiro ficheiro

O último ficheiro em análise denominava-se de mp4-main-multi.mpd que abordamos de seguida, f08e80da-bf1d-4e3d-8899-f0f6155f6efa.mpd, utiliza também o *codec* mp4a.40.2 para o áudio e os seguintes *codecs* vídeo:

Id	Codec	Bandwidth (bits/s)	Player dimensions (px)	Frame Rate	Audio Sampling Rate
h264bl_low	avc1.42c00d	50 877	320x180	25/1	
h264bl_mid	avc1.42c01e	194 870	640x360	25/1	
h264bl_hd	avc1.42c01f	514 828	1280x720	25/1	
h264bl_full	avc1.42c028	770 699	1920x1080	25/1	
aaclc_high	mp4a.40.2	66 378			44100
aaclc_low	mp4a.40.2	19 079			44100

Tabela 3 - representações disponíveis

Contrariamente aos ficheiros previamente abordados, o valor de utilizada pelo áudio pode utilizar dois valores diferentes o que faz com que a escolha do codec vídeo possa assumir diferentes representações. Para os casos que se seguem:

- 320 Kbits/s, a *bandwidth* disponível permite optar pelo id h264bl_mid (melhor qualidade) ou id h264bl_low sendo que qualquer representação áudio possa ser utlizada e, como tal, seleciona-se o id aaclc_high para obter a melhor qualidade
- 500 Kbits/s, o cenário é exatamente o mesmo que para o caso de uma *bandwidth* de 320 Kbits/s
- 1.5 Mbits/s, este cenário permite que seja utilizada qualquer representação de áudio ou vídeo sendo que se deve optar pelo id aaclc_high para áudio e h264bl_full para vídeo de forma a obter a melhor qualidade disponível para ambos os componentes
- 10 Mbits/s, sendo que o valor de largura de banda disponível é superior ao caso anterior de 1.5 Mbits/s, o cenário mantém-se

Relativamente ao *profile file*, é novamente do tipo `urn:mpeg:dash:profile:isoff-main:2011` que se trata de um perfil principal do tipo MPEG-DASH ISO Base media file format.

2) Análise via Wireshark

2.1) Web Player (bitmovin)

No.	Time	Source	Destination	Protocol	Length	Info
4664	16.150667	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2523741 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4665	16.150668	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2525111 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4666	16.153686	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2526481 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4667	16.153687	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2527851 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4668	16.153688	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2529221 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4669	16.154464	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2530591 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4670	16.155182	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2531961 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4671	16.155183	2.16.65.138	192.168.43.188	TLSv1.2	1424	Application Data [TCP segment of a reassembled PDU]
4672	16.157402	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2534701 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4673	16.157403	2.16.65.138	192.168.43.188	TLSv1.2	1424	Application Data [TCP segment of a reassembled PDU]
4674	16.160960	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2537441 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4675	16.160961	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2538811 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4676	16.160962	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2540181 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]
4677	16.165537	2.16.65.138	192.168.43.188	TCP	1424	443 → 56260 [ACK] Seq=2541551 Ack=1825 Win=32768 Len=1370 [TCP segment of a reassembled PDU]

Figura 1 - análise de pacotes de rede

Observando o tráfego pode-se observar que o MPEG-Dash está a operar sobre HTTPS. Com isto conclui-se que o conteúdo que está sendo transmitido entre cliente e servidor está encriptado utilizando TLS sobre uma conexão TCP o que não permite obter mais detalhes relativo à transmissão de dados.

2.2) Cliente local (Shaka Player)

65259	303.500990858	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
66210	308.109011832	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
66212	308.112447811	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
68560	382.031898849	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
68564	382.035525215	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
68652	382.054490884	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1
68656	382.057607828	172.16.20.63	192.168.109.124	HTTP	355	GET /input_low_dash.mp4 HTTP/1.1
68904	382.094613629	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1
68906	382.097502891	172.16.20.63	192.168.109.124	HTTP	355	GET /input_low_dash.mp4 HTTP/1.1
69282	385.146040555	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1

Figura 2 - análise de pacotes de rede

Após a instalação do Shaka Player num dos computadores do laboratório, estabeleceu-se uma conexão com um servidor DASH que não tinha qualquer tipo de encriptação. Assim sendo, conseguimos interpretar melhor o tráfego capturado via *Wireshark* e constatar que a informação era transmitida por pacotes, sendo adaptada conforme a *bandwidth* disponível e os recursos da rede.

3) Implementação de uma estrutura de rede para reprodução de vídeo com interferência via iperf

3.1) Topologia

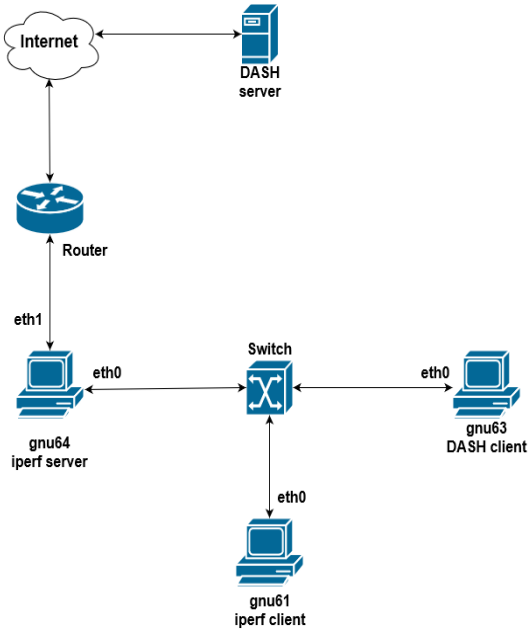


Figura 3 - topologia de rede

A topologia apresentada comporta três computadores, um *switch* e um *router*, este último conectado à Internet. Os computadores gnu61 e gnu63 vão partilhar a mesma VLAN com uma interface do computador gnu64 enquanto que a outra interface se encontra conectada ao router. Desta forma, todo o tráfego de rede terá de ser roteado via gnu64. Esta topologia tem o intuito de fazer com que o tráfego, partilhando a mesma ligação *FastEthernet*, seja roteado todo pela mesma ligação e, com isso, permitir saturar a *bandwidth* utilizando o *iperf*.

3.2) O *iperf*

O *iperf* é um software que permite a injeção de pacotes (TCP ou UDP) de forma a testar a largura de banda e medir o desempenho de certa rede. São necessários dois computadores diferentes, um para atuar como cliente (gnu61) e outro como servidor (gnu64).

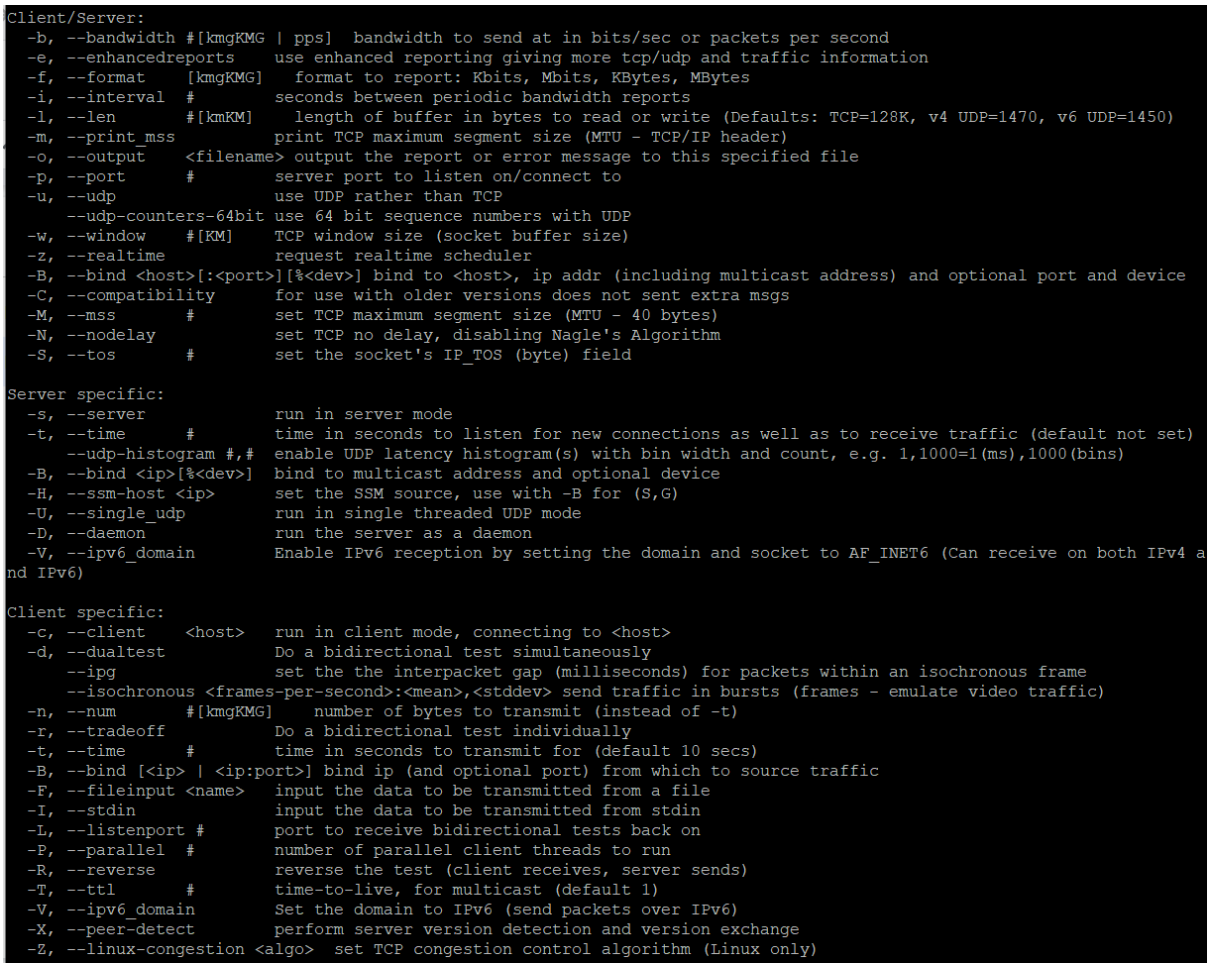


Figura 4 - iperf em bash

Dado que este software corre via terminal, foi necessário especificar os parâmetros do seu funcionamento. Dada a versatilidade do software, há vários parâmetros de funcionamento que podem ser utilizados para melhor cumprir a funcionalidade que desejamos. Desses parâmetros, importante destacar:

- ➔ -s > correr o programa em modo servidor
- ➔ -c > correr o programa em modo cliente
- ➔ -u > correr em modo UDP (restrito ao cliente, o servidor responde conforme os pedidos do cliente)
- ➔ -t > duração de tempo em que vão ser enviados pacotes (default são 10 segundos)
- ➔ -b > bit rate desejado (restrito ao cliente, se valor for 0 então é o máximo possível)
- ➔ -l > comprimento do buffer para leitura/escrita, isto é, determina o tamanho dos pacotes (restrito ao cliente)
- ➔ -R > inverter direção do teste (passa a ser o servidor a enviar pacotes para o cliente)

Com isto em mente, os comandos utilizados foram os seguintes:

- ➔ servidor: `iperf -s`
- ➔ cliente em modo UDP: `iperf -u -c 172.16.22.64 -b 0 -l 1470 -R`
- ➔ cliente em modo TCP: `iperf -c 172.16.22.64 -b 0 -l 1470 -R`

3.2.1) Primeira tentativa: UDP

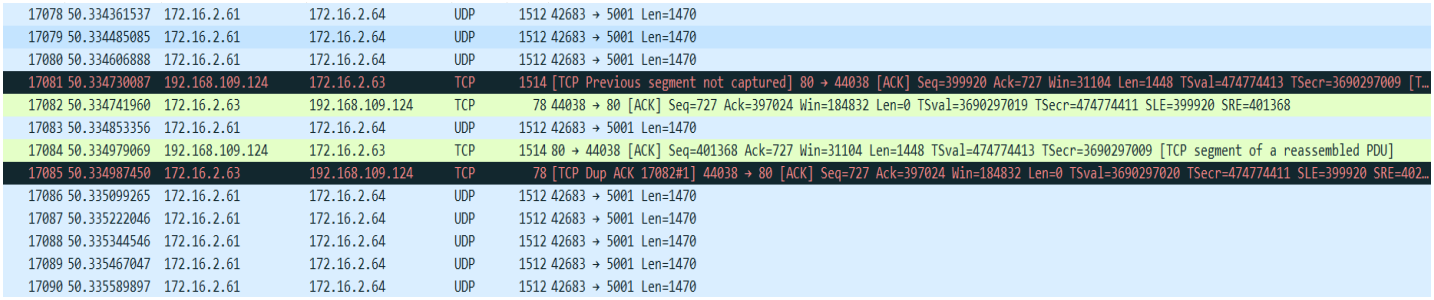
Utilizando o software previamente mencionado, gerou-se uma conexão entre o cliente e servidor *iperf* através dos seguintes comandos:

- ➔ servidor: `iperf -s`
- ➔ cliente: `iperf -u -c 172.16.22.64 -b 70M -l 1470 -R`

Destaca-se que o endereço IP `172.16.22.64` é o endereço do computador onde está a correr o servidor *iperf*.

A cada teste diferente, o *bitrate* era alterado de forma crescente de forma a saturar a ligação e observar o comportamento.

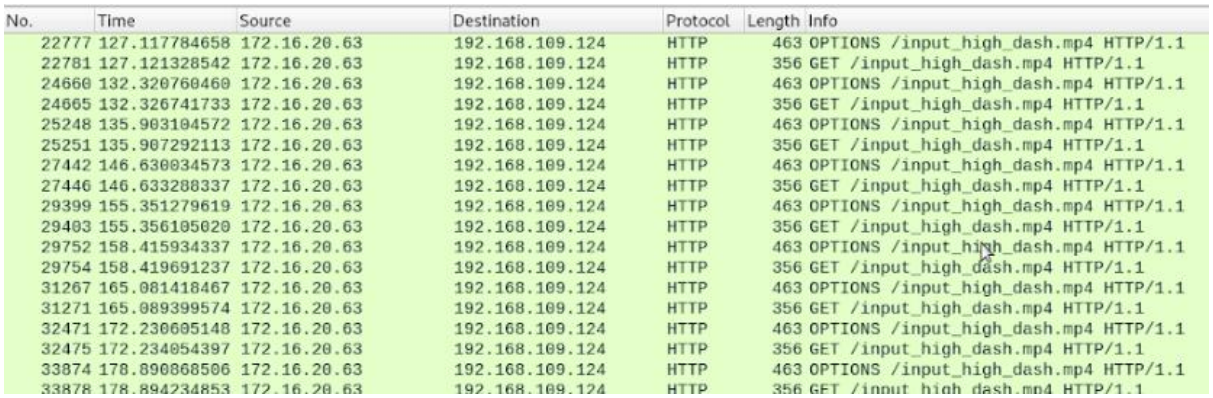
Através do *iperf*, foi criado um fluxo UDP em que em cada iteração aumentava a sua taxa de transferência gerada de maneira a ser possível observar incrementalmente os efeitos do tráfego gerado.



17078	50.334361537	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17079	50.334485085	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17080	50.334606888	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17081	50.334730087	192.168.109.124	172.16.2.63	TCP	1514 [TCP Previous segment not captured] 80 → 44038 [ACK] Seq=399920 Ack=727 Win=31104 Len=1448 TSval=474774413 TSecr=3690297009 [T…
17082	50.334741960	172.16.2.63	192.168.109.124	TCP	78 44038 → 80 [ACK] Seq=727 Ack=397024 Win=184832 Len=0 TSval=3690297019 TSecr=474774411 SLE=399920 SRE=401368
17083	50.334853356	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17084	50.334979069	192.168.109.124	172.16.2.63	TCP	1514 80 → 44038 [ACK] Seq=401368 Ack=727 Win=31104 Len=1448 TSval=474774413 TSecr=3690297009 [TCP segment of a reassembled PDU]
17085	50.334987450	172.16.2.63	192.168.109.124	TCP	78 [TCP Dup ACK 17082#1] 44038 → 80 [ACK] Seq=727 Ack=397024 Win=184832 Len=0 TSval=3690297020 TSecr=474774411 SLE=399920 SRE=402…
17086	50.335099265	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17087	50.335222046	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17088	50.335344546	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17089	50.335467047	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470
17090	50.335589897	172.16.2.61	172.16.2.64	UDP	1512 42683 → 5001 Len=1470

Figura 5 - análise de pacotes de rede

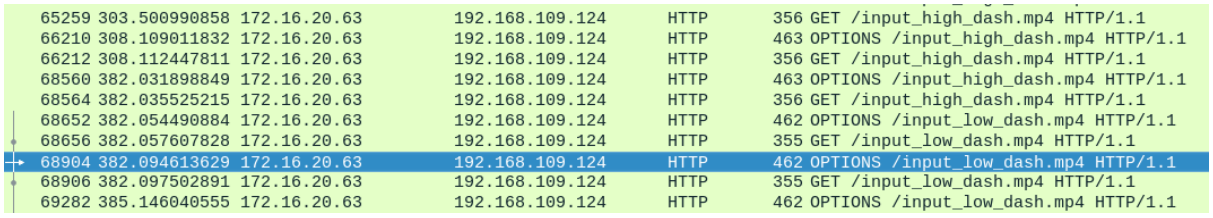
Num primeiro teste, foi escolhido como *bitrate* 70 Mbps, um valor relativamente baixo visto que não afetou o desempenho do cliente DASH que continuou a operar na máxima qualidade.



No.	Time	Source	Destination	Protocol	Length	Info
22777	127.117784658	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
22781	127.121328542	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
24660	132.320760460	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
24665	132.326741733	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
25248	135.903104572	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
25251	135.907292113	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
27442	146.630034573	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
27446	146.633288337	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
29399	155.351279619	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
29403	155.356105020	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
29752	158.415934337	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
29754	158.419691237	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
31267	165.081418467	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
31271	165.089399574	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
32471	172.230605148	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
32475	172.234054397	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
33874	178.890868506	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
33878	178.894234853	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1

Figura 6 - análise de pacotes de rede

No teste seguinte, o comando utilizado foi `iperf -u -c 172.16.22.64 -b 0 -l 1470 -R` o que obrigou a utilizar o maior *bitrate* disponível. Os resultados foram visíveis pois a qualidade da transmissão diminuiu.



65259	303.500990858	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
66210	308.109011832	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
66212	308.112447811	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
68560	382.031898849	172.16.20.63	192.168.109.124	HTTP	463	OPTIONS /input_high_dash.mp4 HTTP/1.1
68564	382.035525215	172.16.20.63	192.168.109.124	HTTP	356	GET /input_high_dash.mp4 HTTP/1.1
68652	382.054490884	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1
68656	382.057607828	172.16.20.63	192.168.109.124	HTTP	355	GET /input_low_dash.mp4 HTTP/1.1
68904	382.094613629	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1
68906	382.097502891	172.16.20.63	192.168.109.124	HTTP	355	GET /input_low_dash.mp4 HTTP/1.1
69282	385.146040555	172.16.20.63	192.168.109.124	HTTP	462	OPTIONS /input_low_dash.mp4 HTTP/1.1

Figura 7 - análise de pacotes de rede

3.2.2) Segunda tentativa: TCP

Seguindo a mesma lógica anterior, procedeu-se exatamente ao mesmo teste mas desta vez utilizando o comando `iperf -c 172.16.22.64 -b 0 -l 1470 -R` de forma a correr em modo TCP.

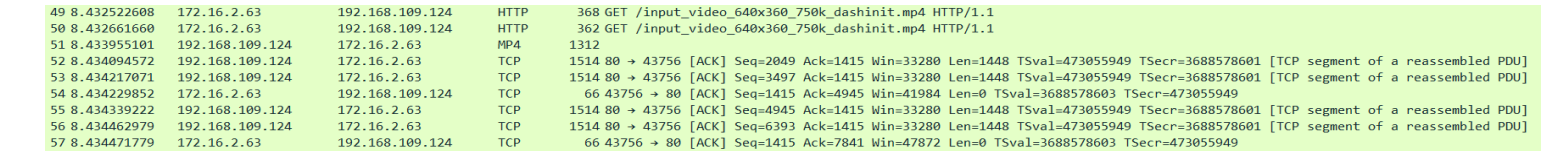


Figura 8 - análise de pacotes de rede

Observa-se que há uma quebra na transmissão de tráfego HTTP relativa ao cliente DASH visto que há uma saturação total da transmissão derivado ao tráfego que o *iperf* está a gerar e a ocupar a maior *bandwidth* disponível. A qualidade do vídeo passa a ser a menor possível e só é possível a sua reprodução pela forma como o Shaka Player opera em que há o carregamento prévio de segmentos futuros do vídeo a ser reproduzido e permite a sua reprodução mesmo enquanto não houver ligação.

4) Controlo de tráfego via TC

A ferramenta TC (Traffic Control) é um mecanismo de QoS (Quality of Service) utilizado em ambiente Linux que procede à configuração de estruturas do kernel essenciais para o controlo de tráfego.

Do ponto de vista do utilizador, é capaz de configurar filas de espera e associar classes a cada uma delas, assim como configurar filtros. De forma a comunicar com as funções de rede do kernel, sockets netlink são utilizados.

Para invocar o mesmo, o seguinte comando é utilizado:

```
➔ tc [ OPTIONS ] OBJECT { COMMAND }
```

1. OPTIONS, onde uma miríade de parâmetros pode ser invocada nomeadamente:
    a. -stats, que garante estatisticas relativas ao objecto
    b. -details, incrementa a resolução na numeração dos objectos
    c. -version, mostra informações relativas à versão do comando sendo utilizada

d. -help, informações relativas à sintaxe
e. -raw, demonstra a forma como os comandos invocados pelo utilizador sao interpretados pelo kernel

2. OBJECT, representa o objeto alvo da configuração. Podem ser do tipo:
   a. filter, para a configuração de filtros
   b. qdisk, para a configuração de disciplinas de serviço
   c. class, para a configuração de classes
3. COMMAND, específico para cada objecto

TC Filter

Recorre-se a filtros para classificar os pacotes através do campo próprio de cada pacote. Estes filtros são invocados aquando da disciplina de serviço a qual se vai associar os pacotes de entrada a uma das suas classes de serviço.

A criação de um filtro é feita da seguinte forma:

➔ `tc filter [COMMAND] dev STRING [ root | parent CLASSID ] [ preference PRIO ] [protocol PROTO ] [estimator INTERVAL TIME_CONSTANT] [ handle FILTERID ][ [ FILTER_TYPE ] [ help | OPTIONS ] ]`

1. Neste cenário, o campo **COMMAND** pode assumir:
   - add, adicionar nova disciplina de serviço à hierarquia
   - delete, apagar por completo uma hierarquia
   - change, alterar parâmetros duma disciplina de serviço.

2. Para identificar a interface de rede que opera sobre a hierarquia, deve-se usar o parâmetro "**dev**"
3. O campo **CLASSID** representa a disciplina de serviço ou a classe em que o filtro está istalado, e pode ser do tipo:
   a. root, diretamente na raiz
   b. parente, o default

4. O campo preference **PRIO** serve para identificar o filtro e determinar a ordem de aplicação dos vários filtros. Filtros distintos não podem ter a mesma prioridade nem mesmo protocolo.
5. O campo **PROTO** especifica o protocolo ao qual o filtro vai ser aplicado.
6. O campo **ESTIMATOR** é utilizado no caso de se querer instalar um estimador. Desta forma é possível ao filtro exercer funções de policiamento.
7. O campo **FILTERID** é um identificador único do filtro e permite identificar o elemento do filtro a manipular. O formato do filtro varia de acordo com o classificador.

TC QDISC

"Queuing Disciplines" é na verdade um scheduler. Há a necessidade de configurar cada interface de saída com um de forma a tratar os pacotes que entram numa fila. O scheduler padrão no Kernel é do tipo First In First Out (FIFO), ainda que cada scheduler possua a sua forma própria de endereçar os pacotes.

A criação de um filtro é feita da seguinte forma:

➔ `tc qdisc [COMMAND] dev STRING [ handle QHANDLE ] [ root | ingress | parent CLASSID ] [ estimator INTERVAL TIME_CONSTANT ] [ [ QDISC_KIND ] [ help | OPTIONS ] ]`

Aos parâmetros descritos anteriormente, acrescenta-se agora:

1) Um identificador único de classe e hierarquia qdisc no interior da estrutura de controlo designado por QHANDLE
2) Um representante do tipo de tratamento de dados associado à fila de espera (ex. FIFO, TBF.PRIO, CBQ, RED) designado por QDISK_KIND

TC Class

Há uma atribuição de uma disciplina de fila de espera a cada classe de. Quando há a chegada de pacotes a uma fila de espera, o qdisc aplica filtros de forma a determinar a que classe cada pacote pertence. Aí, é invocada a disciplina da fila-de-espera correspondente, que fica responsável pela aplicação das regras associadas à entrada e saída de pacotes da fila.

Há duas formas para se identificar uma classe. Ou através do identificador de classe definido pelo utilizador ou pelo uso de um identificador interno associado pelo kernel a cada classe. Estes identificadores são únicos e são atribuídos pelo qdisc.

A sintaxe do comando é a seguinte:

➔ `tc class [COMMAND] dev STRING [ classid CLASSID ] [ root | parent CLASSID ] [ [ QDISC_KIND ] [ help | OPTIONS ] ]`

Os parâmetros são idênticos ao qdisk excetuando em relação ao identificador da classe que é hierárquico do tipo superior:inferior, apesar de que, dado que a parte superior é idêntica à do identificador da disciplina do serviço, a sua omissão é opcional.

4.1) Aplicação

Visto que o principal deste trabalho é adaptar o trafego para uma taxa especifica de forma simples e eficaz, iremos optar pela hierarquia HTB (Hierarchical Token Bucket) que permite uma fácil garantia na alocação da largura de banda e a definição dos limites superior na partilha entre classes.

Com base no mencionado anteriormente, os parâmetros utilizados nos comandos foram:

1) qdisc > `tc class tc qdisc add dev eth0 root handle 1: htb default 12`
2) class
   a. `tc class add dev eth0 parent 1: classid 1:1 htb rate 100 mbit ceil 100mbit`
   b. `tc class add dev eth0 parent 1:1 classid 1:11 htb rate 70mbit ceil 100mbit`
   c. `tc class add dev eth0 parent 1:1 classid 1:12 htb rate 30mbit ceil 100mbit`
3) filter > `tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip protocol 17 0xff flowid 1:11`

Num primeiro passo procedeu-se à definição dum qdisc HTB na raiz de eth0, especificando que tráfego não caracterizado é à classe 1:12, com identificador da raiz 1.

Numa fase seguinte, estabeleceu-se uma classe descendente da raiz e correspondente à Ethernet, permitindo a permutação de tokens através do comando ceil. A classe previamente mencionada subdivide-se agora em Classe 1:11 para tráfego UDP, com *bitrate* de 70 Mbps e em classe 1:12 com *bitrate* de 30 Mbps.

No final pôde-se concluir que quando não há qualquer fluxo UDP presente, os restantes partilham a largura de banda entre si independentemente das regras estabelecidas. O fluxo do cliente DASH, que utiliza os restantes 30 Mbps disponíveis, não sofre interferência por parte desse fluxo visto que os tráfegos TCP e UDP não conseguem interferir com o cliente DASH.

5) Conclusão

A realização deste trabalho ajudou-nos a adquirir novas competências no âmbito de *adaptative streaming* em especial o MPEG-DASH. Há agora uma melhor compreensão na forma como a reprodução do conteúdo multimédia é efetuada, assim como a mesma reage face a interferências externas. Houve uma introdução ao *iperf* e de como o podemos usar para simular situações que possam pôr em causa o funcionamento dum serviço e, daí, desenvolver soluções que possam colmatar essas falhas. Num ponto de vista menos concreto, houve também uma aprendizagem mais exaustiva relativa ao *kernel* do Linux nomeadamente uma demonstração de como o mesmo opera em diferentes cenários de manutenção de tráfego.