U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Lab 3

Provide QoS to Adaptive Video

Rita Martinho
up201709727@fe.up.pt

Gonçalo Xavier
up201604506@fe.up.pt

Dezember 2020

Part 1: MPD file analysis

ID	Codec	Bandwidth	Segment Size	Player Dimensions	Frame Rate	Audio Sampling Rate
h264bl_low	avc1.42c00d	50877	10s	320 x 180	25:1	-
h264bl_mid	avc1.42c01e	194870	10s	640 x 360	25:1	-
h264bl_hd	avc1.42c01f	514828	10s	1280 x 720	25:1	-
h264bl_full	avc1.42c028	770699	10s	1920 x 1080	25:1	-
aaclc_low	mp4a.40.2	19079	9,52s	-	-	44100
aaclc_high	mp4a.40.2	66378	9,52s	-	-	44100

Figure 1: mp4-main-multi-mpd-AV-NBS file analysis

The table above summarizes the information obtained from the "mp4-main-multi-mpd-AV-NBS.mpd" file and will serve as reference to answer the questions bellow.

Q: How many video and audio codecs are available?
There are 4 video codecs: avc1.42c00d, avc1.42c01e, avc1.42c01f, avc1.42c028, and 2 audio codecs: mp4a.40.2 appearing in both aaclc_low, aaclc_high.

Q: Which is the segment size?
The duration of each segment is present on the *SegmentList* tag of its *Representation*:

```
<SegmentList timescale="1000" duration="10000">
```

With *"timescale"* being defined as a number of units per second. As such, video segments are 10s long, while audio segments are 9.52s.

The *"Period"* tag also indicates that it's total *"duration="PT0H10M0.00S"*, (10 minutes), which adds up with the mentioned segment size given that each *"SegmentList"* is made up of 60 *"Segments"*.

Q: Which audio and video data rates are available?

In each *Representation* tag, the *"bandwidth"* field declares the necessary data rate for continuous playout of its Segments.

That being said, the available data rates are:

- Video:
  - h264bl_low: 50877 bps
  - h264bl_mid: 194870 bps
  - h264bl_hd: 514828 bps
  - h264bl_full: 770699 bps
- Audio:
  - aaclc_low: 19079 bps
  - aaclc_high: 66378 bps

Q: Which options would you have for an available bandwidth of 500kbps, 1.5Mbps, 10Mbps?

For convenience, we will refer to the segment ID's by video_low/_mid/_hd/_full and audio_low/_high.

With an available bandwidth of 500kbps the best option would be: **audio_high + video_mid** = 66kbps + 195kbps.

With a bandwidth of 1.5Mbps the best option possible would be the best one offered: **audio_high + video_full** = 66kbps + 770kbps.

Likewise, with 10Mbps available the same thing happens.

Q: Play the video https://dash.akamaized.net/akamai/bbb_30fps/bbb_30fps.mpd, observe what happens for different controls (adaptation with different algorithms). Use Wireshark to observe the playing of the video stream. Describe the data transfer profile of the video stream, preferably with a plot.

A similar analysis of the bbb_30fps.mpd file was made, resulting in the table of figure 2. For convenience sake, given that all codec's ID start with "bbb_30fps_", they will be reference only by the later part of their name.

ID	Codec	Bandwidth	Segment Size	Player Dimensions	Frame Rate	Audio Sampling Rate
320x180_200k	avc1.64000d	254320	4s	320 x 180	30:1	-
320x180_400k	avc1.64000d	507246	4s	320 x 180	30:1	-
480x270_600k	avc1.640015	759798	4s	480 x 270	30:1	-
640x360_800k	avc1.64001e	1013310	4s	640 x 360	30:1	-
640x360_1000k	avc1.64001e	1254758	4s	640 x 360	30:1	-
768x432_1500k	avc1.64001e	1883700	4s	768 x 432	30:1	-
1024x576_2500k	avc1.64001f	3134488	4s	1024 x 576	30:1	-
1280x720_4000k	avc1.64001f	4952892	4s	1280 x 720	30:1	-
1920x1080_8000k	avc1.640028	9914554	4s	1920 x 1080	30:1	-
3840x2160_12000k	avc1.640033	14931538	4s	3840 x 2160	30:1	-
bbb_a64k	mp4a.40.5	67071	4s	-	-	48000

Figure 2: bbb_30fps file analysis

10 video codecs are available, with different bandwidths, all with segment sizes of 4s (duration: 120 on a timescale of 30):

```
<SegmentTemplate duration="120" timescale="30"
media="$RepresentationID$/$RepresentationID$_$Number$.m4v"
startNumber="1" initialization="$RepresentationID$/$RepresentationID$_0.m4v"/>
```

A single audio codec is available, also with segment size of 4s (duration: 192512, on a timescale of 48000).

```
<SegmentTemplate duration="192512" timescale="48000"
media="$RepresentationID$/$RepresentationID$_$Number$.m4a"
startNumber="1" initialization="$RepresentationID$/$RepresentationID$_0.m4a"/>
```

Given that only 1 audio codec is available, only the image quality fluctuates in congestion scenarios. The maximum viewing experience, video codec 3840x2160 with bbb_a64k audio, requires a bandwidth of 14998609bps≈**15Mbps** .

TEMOS DE TENTAR PERCEBER, QUE FOI UMA COISA QUE A PROF FALOU, SE A CADA SEGMENTO O DASH SERVER RESPONDE COM DADOS, OU SE RECEBE VÁRIOS SEGMENTOS E SÓ DEPOIS RESPONDE COM DADOS, SE ESTES DADOS TÊM SEMPRE O MESMO TAMANHO...

Part 2: Bottleneck for quality degradation experiment

In order to force the DASH traffic to flow on a bottlenecked connection, an Iperf Server and Client were implemented within the following architecture:

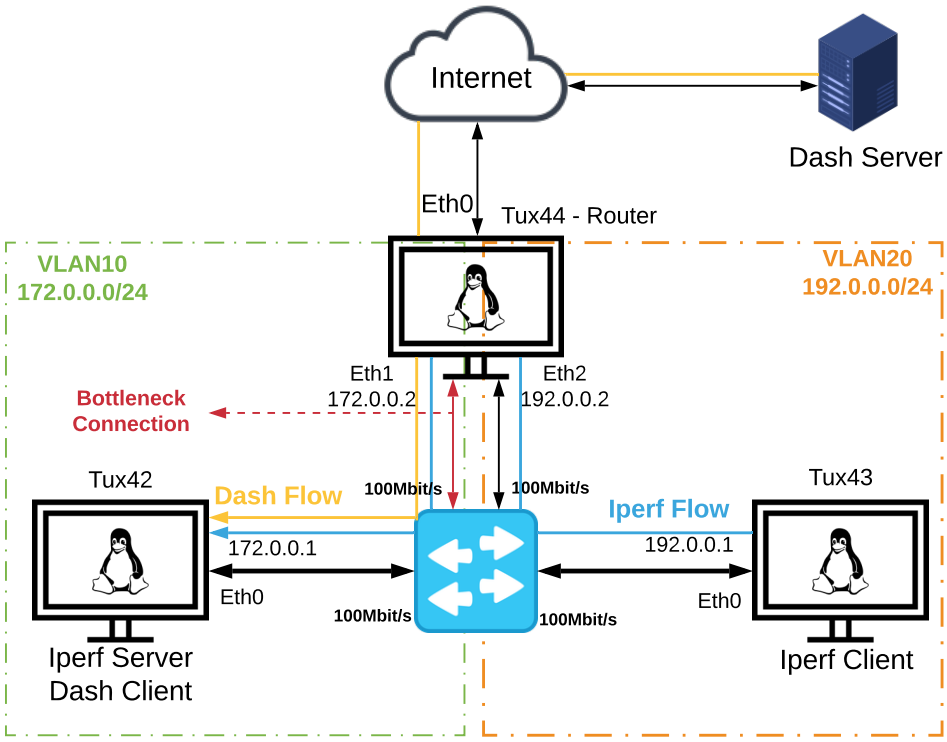

Figure 3: Implemented Iperf and Dash architecture

Iperf traffic flows from client to server, and since Tux43 and 42 are located in different VLANs, this traffic needs to go through Tux44 (acting as a router).

Since Tux44 also functions as an outside router to the general Internet, Dash traffic (from Server to Client) is also forwarded through Tux44's Eth1, throttling the egress queue in Tux44's Eth1 connection.

Iperf traffic was generated on it's client side, **-c**, to the Iperf server (**172.0.0.1**) using the following command:

```
iperf3 -c 172.0.0.1 -t 0 -b 90M
```

To note both the **-t 0** to ensure an infinite flow and **-b 90M** to set a 90Mbit/s bandwidth.

On the Iperf server side a simple command was used to specify it as a server:

```
iperf3 -s
```

Is possible to observe the degradation of the DASH stream by simultaneously having a Iperf stream (from Tux43 to Tux42) and a DASH stream active:

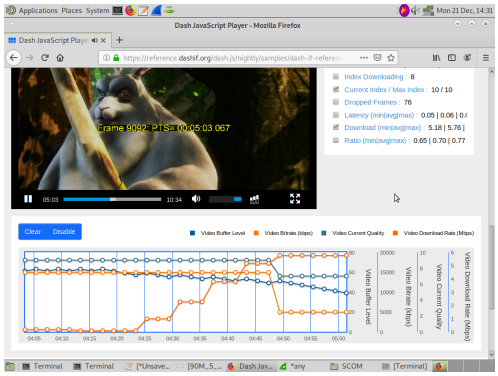

Figure 4: Stream Quality: 10/10 - No Iperf

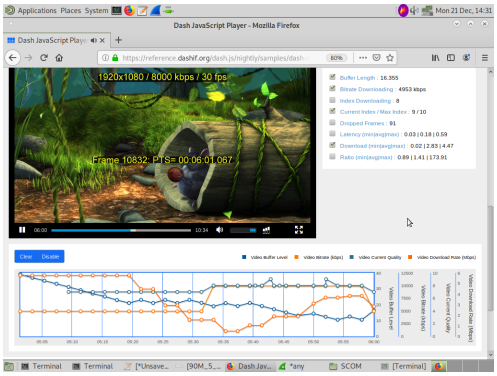

Figure 5: Stream Quality: 8/10 - 90M Iperf

With an active **UDP** Iperf stream, the DASH player behaved as follows:

"Finish the iperf UDP traffic before the end of the stream, and observe what happens."

"Show your observations using the wireshark trace analysis tool, mark the timestamps at which you added and changed cross traffic. Justify what you are seeing."

Q: Observe the behaviour of the video player:

- **Q: Which data rates is the player choosing?**

  Without a UDP Iperf stream the player chooses the maximum quality available: **10/10 - 3840x2160 - 14931kbps**.

  With UDP cross traffic, these data rates decrease, as does the video quality, to a resolution of **9/10 - 1920x1080 - 9914kbps**, and sometimes even **8/10 - 1280x720 - 4952kbps**.

  Note that, as showed in figure 6, even with Iperf cross traffic, the DASH client still adapts when possible (increasing it's quality from 8 to 9).

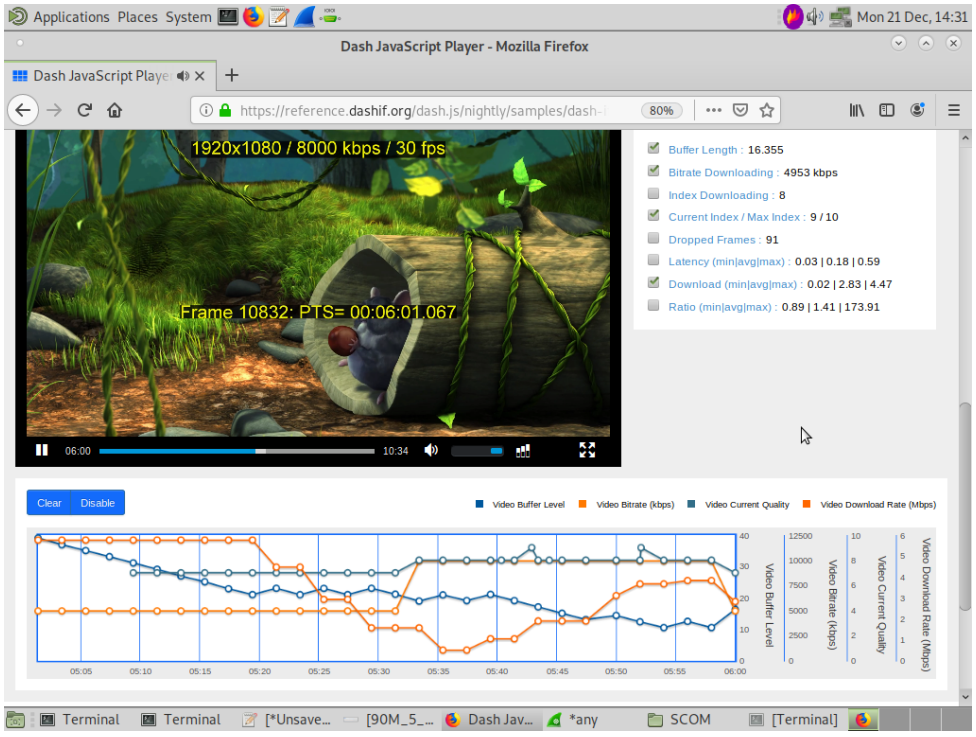

Figure 6: DASH client quality increasing from 8 back to a 9

- **Q: How often are segments requested?** ¡++¿
- **Q: How does the player tell the server which rate to use?** ¡++¿

With an active **TCP** Iperf stream, the DASH player behaved as follows:

"Show your observations using the wireshark trace analysis tool, mark the timestamps at which you added and changed cross traffic. Justify what you are seeing."

"Observe the behaviour of the video player."

- Which data rates is the player choosing?"
- "How often are segments requested?":
- "How does the player tell the server which rate to use?"

Q: Were does the behaviour differer for TCP and UDP cross traffic? How? Why? Did you expect this behaviour?

Given TCP's congestion avoidance and it's congestion window scheme, a different behaviour for it's cross traffic was expected. Whilst UDP's cross traffic simply tries to occupy the entire available bandwidth, with no regard to traffic congestion, TCP's cross traffic "backs-off" once it reaches a timeout packet and enters *slow-start*.

The impact of this in the DASH stream is minimal, even though the DASH client is aware of the TCP cross traffic (rise of the video download rate), no actual codec change is registered. Hence, the DASH client quality is not affected by Iperf's TCP cross traffic.

Such can be confirmed by the figure 7, where incoming Iperf traffic throughput oscillates, due to TCP's "back-off" between 45-90Mbit/s, not being consistently high enough to compromise DASH's quality.

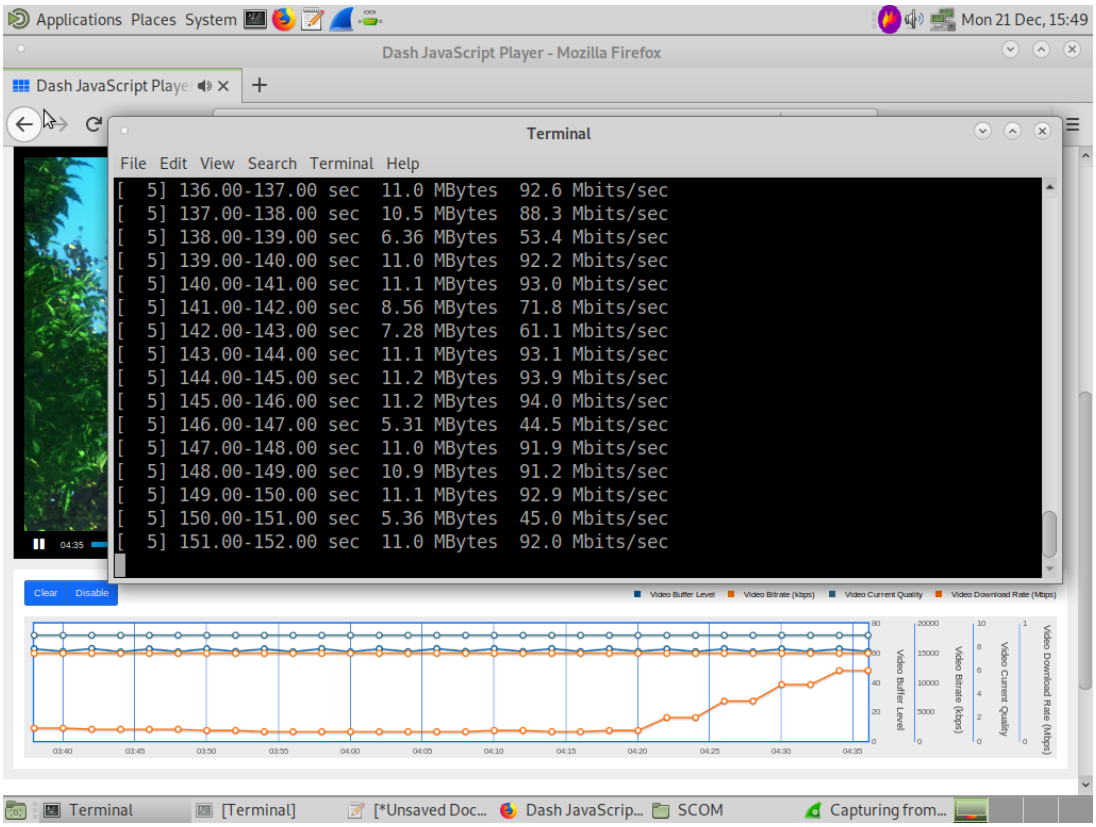

Figure 7: Iperf Incoming Traffic and DASH Client impact during both flows

Part 3: Bandwidth reservation for video flow

In order to safeguard the DASH flow from degradation due to bandwidth sharing, a HTB (Hierarchy Token Bucket) queueing policy was implemented on Tux44, reserving a minimum guaranteed bandwidth for DASH traffic.

Using the Linux traffic control tool (**tc**) a HTB queueing discipline with the handle 1: was attached to Tux44's Eth1:

```
tc qdisc add dev eth1 root handle 1: htb default 12
```

A general overview of this hierarchy queueing discipline can be seen on figure 8.

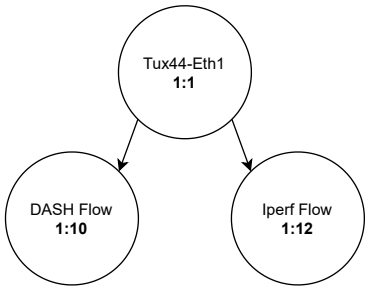

Figure 8: Implemented HTB Hierarchy on TUX44's Eth1

3 TC classes were created for the different types of traffic and their requirements:

```
tc class add dev eth1 parent 1: classid 1:1 htb rate 100mbit ceil 100mbit
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 20mbit ceil 100mbit
tc class add dev eth1 parent 1:1 classid 1:12 htb rate 80mbit ceil 80mbit
```

(rate: guaranteed available bandwidth for that class;
ceil: highest rate a class can reach, if it's parent has bandwidth to spare).

The 1st class is considered a root class, with the following ones being it's leafs. Class 1:10 (DASH flow) has a guaranteed bandwidth of 20mbit but can reach 100mbit if the root class has bandwidth to spare while Class 1:12 (default and Iperf) has a guaranteed and maximum bandwidth of 80mbit.

A value of 20mbit was chosen for the DASH traffic, since it's maximum codec only requires 15mbit.

To associate the correct packets to their traffic classes, the following filter was applied:

```
tc filter add dev eth1 protocol ip parent 1:0 prio 1 u32 match ip src 194.210.238.81 flowid 1:10
```

Associating the packets on Tux44's Eth1, with a source IP of 194.210.238.81 (DASH Server's IP) to the class 1:10 and it's requirements.

Q: Re-run the experiments with cross traffic that you performed before, document them using the TCP throughput plots, and explain the observations.

With this HTB queueing policy in place the DASH flow is never compromised, as depicted in figure 9 regardless of the protocol of the Iperf generated traffic (TCP or UDP).

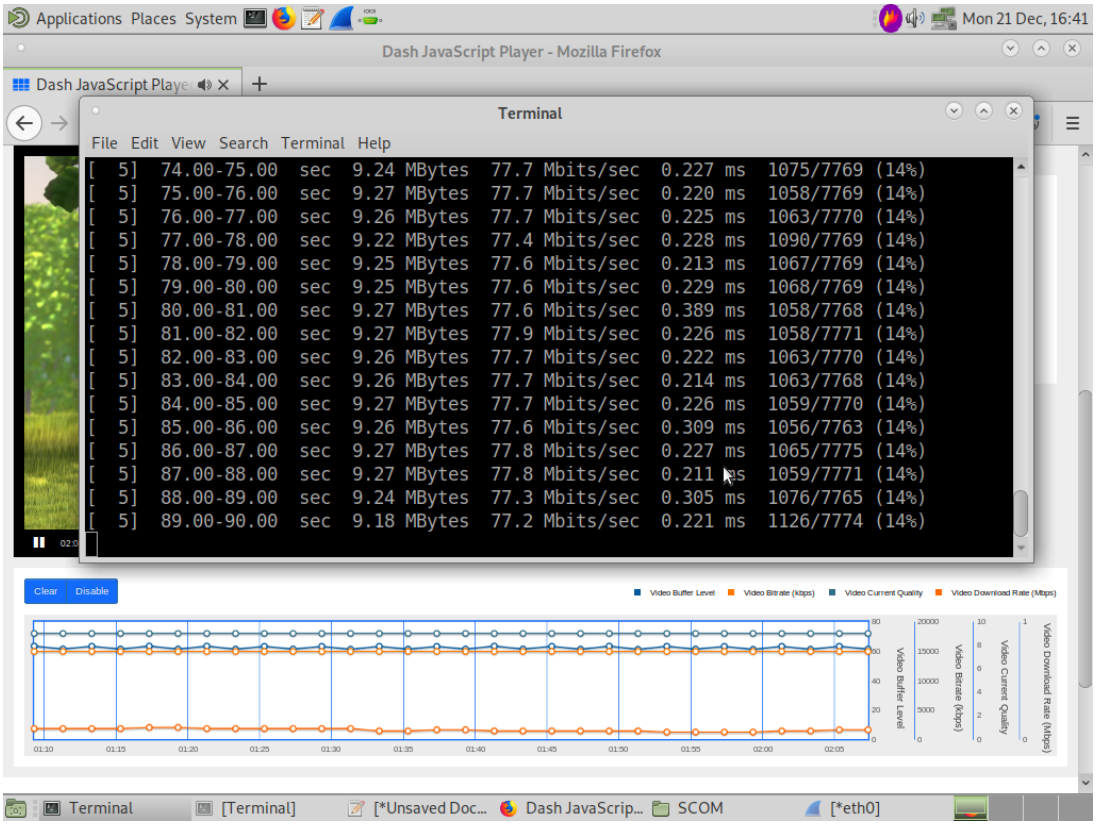

Figure 9: DASH client constant quality with UDP Iperf traffic active, thanks to HTB policy

The differences between both protocols, on the Iperf client side, still remained, with the aforementioned TCP congestion control mechanisms still present, as depicted on figure 10, and the immediate and persistent total bandwidth usage of UDP, as seen on figure 11.

Note that Iperf's client sends 90Mbit, and not the imposed limit of 80M since the this limitation is only present on the egress Tux44's Eth1 connection, and not on the Iperf client itself. Again, that limitation can be seen on the server's side on figure 9.

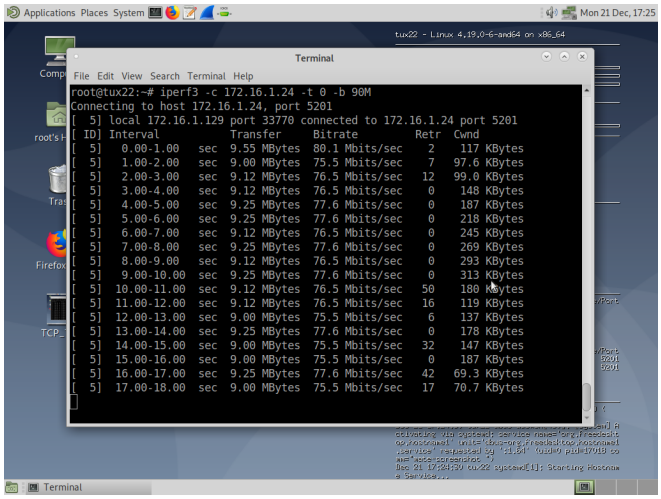

Figure 10: TCP Iperf traffic with HTB policy

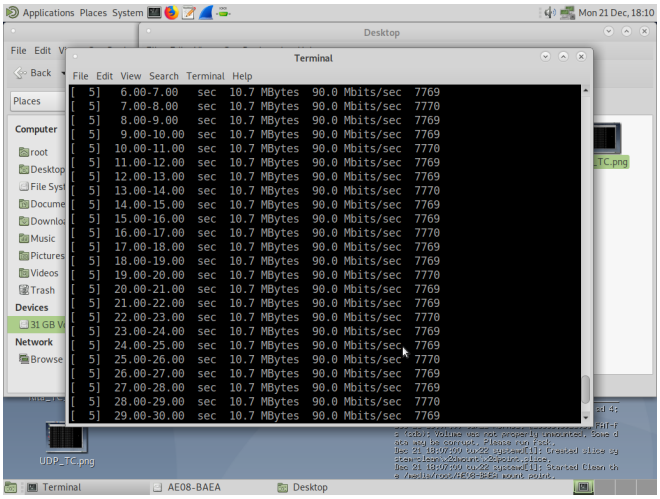

Figure 11: UDP Iperf traffic with HTB policy

FOTO DAS ESTATISTICAS, A REFERIR AS CENAS MAIS EVIDENTES

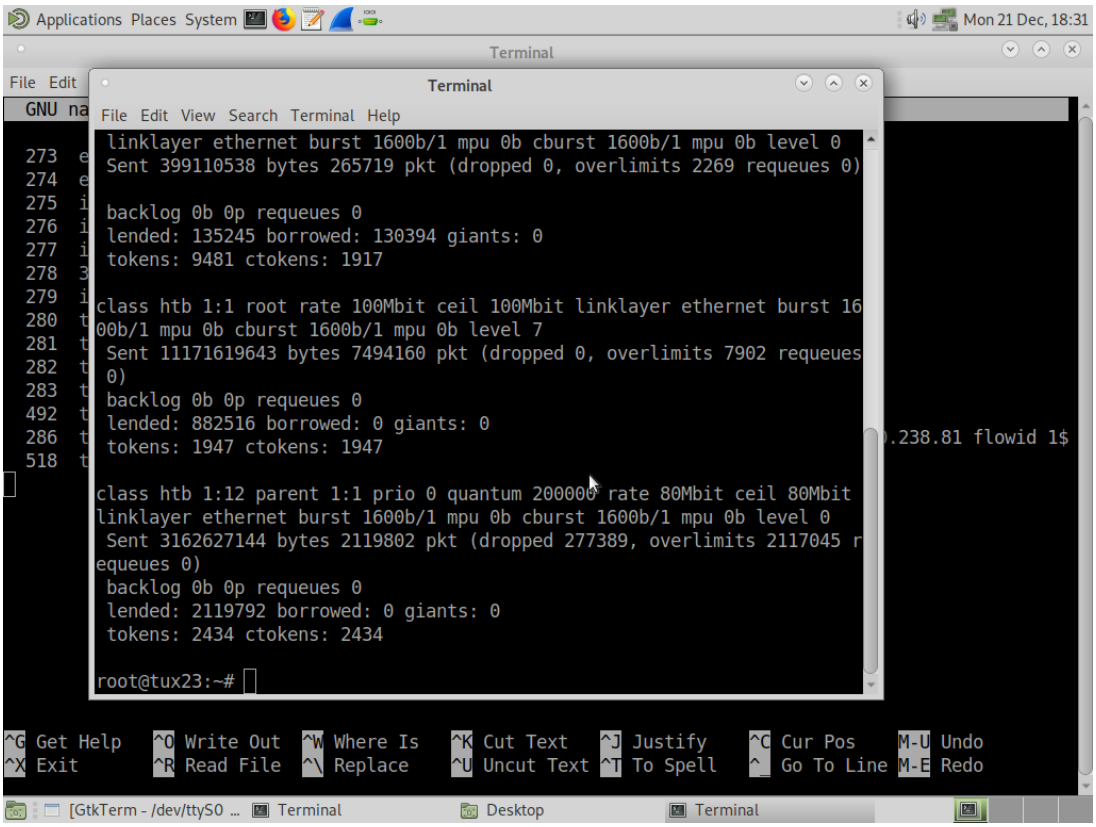

Figure 12: TC stats - output of: tc -s -d class show dev eth1