Internet Performance and Troubleshooting Lab

Video streaming

Report 5 **Group 9**

Giuseppe Bruno	s319892
Michela Salvadori	s302544
Salvatore Giarracca	s314701

February 26, 2024

1 Introduction

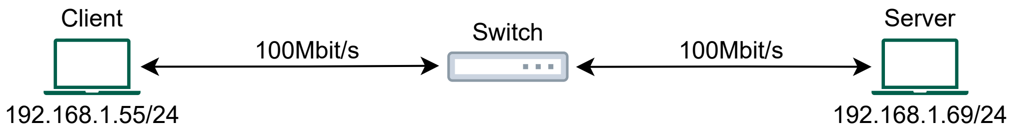


Figure 1: testbed

1.1 Environment setup

The testbed is shown in Figure 1. Using the **qdisc** tool, a **delay of 200** ms was introduced, both client-side and server-side (RTT = 400ms) and additionally **all the offloading** capabilities of the NICs have been **disabled**. This is the base setup for all the following experiments. Real-time Transport Protocol (RTP) was not discussed (relies on UDP under the hoods).

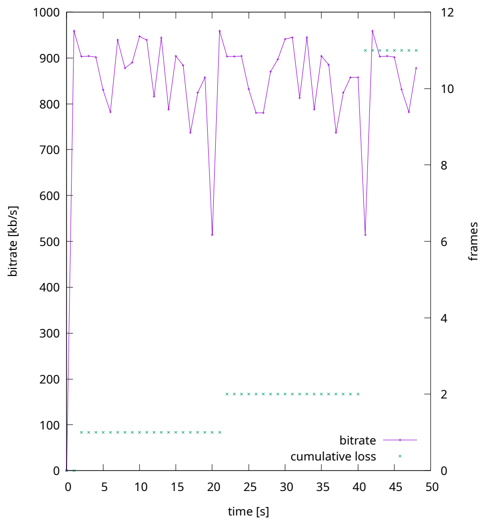
2 UDP/HTTP performance comparison

Due to the limited power of the server (dual core cpu), we opted to previous transcode the video sample and then launch the stream.

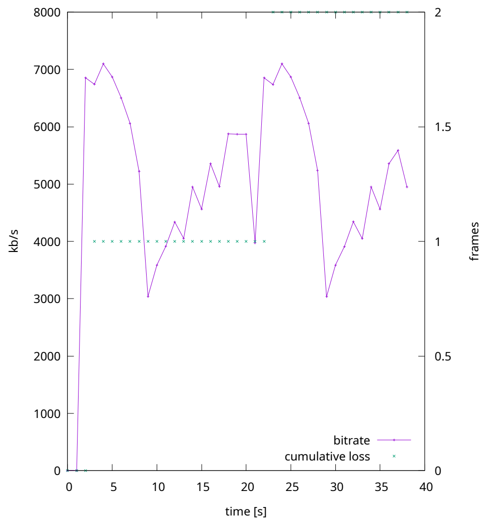
```
cvlc input.mkv --sout="#transcode{vcodec=h264,vb=BITRATE} : file{mux=mp4,dst=output.mp4}"
```

2.1 UDP

Figure 2 shows the differences between two different encoding bitrate of the same video sample (**jellyfish-50-mbps-hd-h264**). In particular in Figure 2a the video is encoded at 800 Kbits/s, while in Figure 2b the video is encoded at 6.4 Mbits/s. In the both cases, no issues rises during the streaming due to the higher bandwidth available compared to the one needed for the streaming operation. The only important difference to notice is the video quality between the two encode types: the higher is the encoding bitrate, the better will be the quality of the video.



(a) bitrate 800

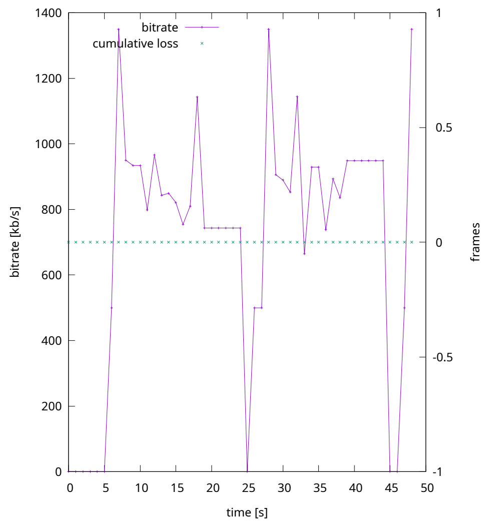


(b) bitrate 6400

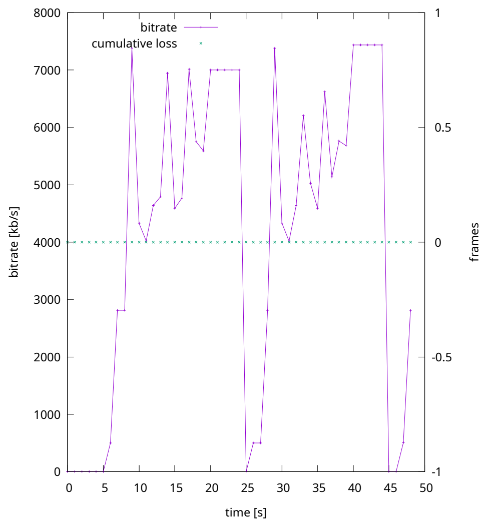
Figure 2: UDP

2.2 HTTP

When HTTP protocol is used to perform the video streaming, there are no differences from the point of view of video quality. The stream goes smoothly as in the UDP case. The difference this time is due to the use of TCP protocol under the hood. Thus, as it is possible to see in Figure 3, no packet losses are registered during the entire duration of the streaming.



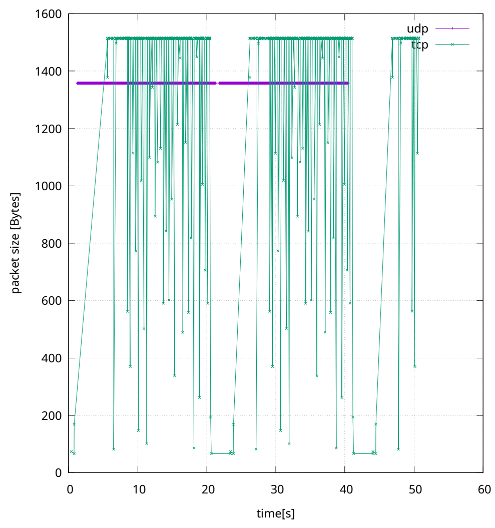
(a) bitrate 800 Kbits/s



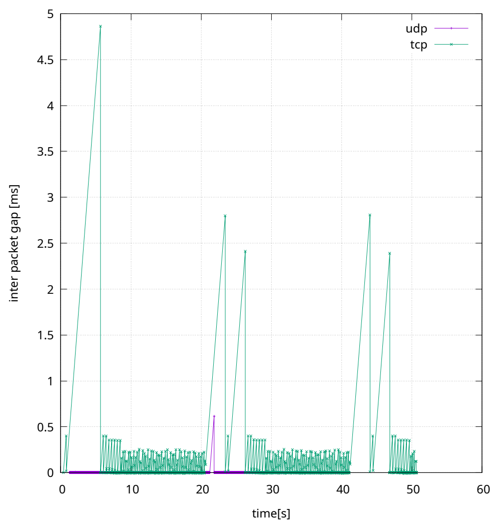
(b) bitrate 6.4 Mbits/s

Figure 3: HTTP

2.3 Evaluating UDP/TCP packet size and inter packet gap



(a) Packet size



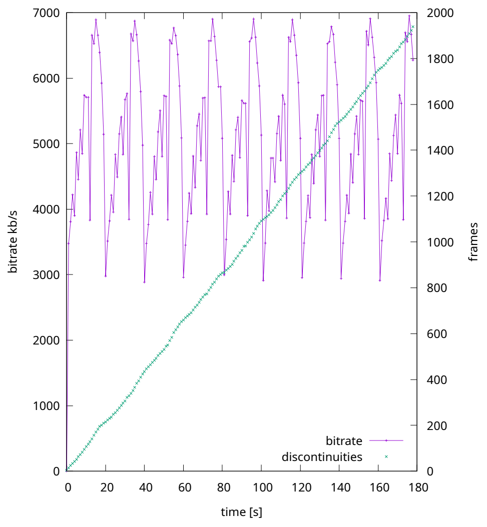
(b) Inter packet gap

Figure 4: UDP/TCP comparison

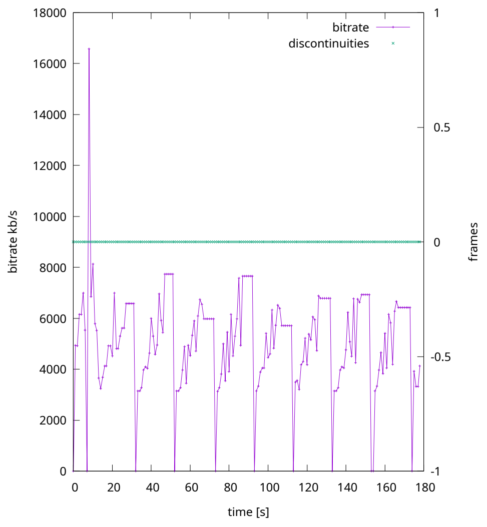
Figure 4a shows how the **packet size** vary when UDP or TCP are used. In case of TCP, it changes frequently compared to UDP: this is due to the presence of sophisticated control mechanism offered by TCP and it is often equal to the maximum size (**1514B**) as we already discussed in previous labs. On the other hand, when using UDP the packet size is always equal to **1358B**. This seam to be strange because UDP has a smaller header compared to TCP one. Investigating a bit, we discovered that, when video streaming is performed with UDP, **MPEG Transport Stream** is used. This is a particular format in which the data stream is split and it's basic unit is the "**packet**" which is 188 Byte long. MPEG TS is needed because in UDP there is no concept of order. Theoretically, an UDP datagram could have 1480B payload, but due to this structure "packet oriented", only 7 * 188B = 1316B are encapsulated inside an UDP datagram. Figure 4b shows the **time gap** between the receipt of the current packet and the previous one in both TCP and UDP cases. In particular we can see how this gap is higher when TCP is used, reaching peaks of almost 5 ms and 2.5 ms when the video is restarted, while when UDP is used, this gap is more stable with less variations and in the order of μs.

3 Impact of packet loss probability

In this experiment, in addition to the base setup, we added **3%** of packet **loss probability** only server side. We also tried only client side, but no differences were recorded, compared to the previous experiment. However, here the sample video encoded at 6.4 Mbits/s is used in both UDP and TCP stream.
VLC can drop packets because of two reasons: (1) if the hardware (client) can't decode fast enough, VLC will skip (drop) them, and they will appear in the statistics (**lost**). Packets are organized with timestamps, so the vlc player can decode them in order. (2) If the packet has an invalid timestamp, or its timestamp is below the current timestamp, VLC also drop it (**discontinued**). When we first considered the cumulative losses, we noticed that they are less than the previous experiment. It could be that now we have to consider two probabilities: the first is the "natural" probability to lose a datagram (due to the use of UDP) and in addition the "artificial" one added by using **qdisc**. For this reason we considered the **discontinued** packets instead of **lost** packets taken from cli statistics. As we can see in Figure 5a, the number of discontinued packets reaches almost 2000 in case of UDP (only 2 when 0% ploss). On the other hand, when we consider TCP, Figure 5b, we have always 0 discontinued packets (and also lost ones) due to reliability of TCP.



(a) UDP



(b) TCP

Figure 5: Impact of 3% probability loss

As a result, in the UDP streaming the video was completely not enjoyable and most of the times we seen completely corrupted frames. In the HTTP streaming, instead, the video was smooth as in the first experiment without noticing any particular issue during reproduction.

4 Impact of channel capacity

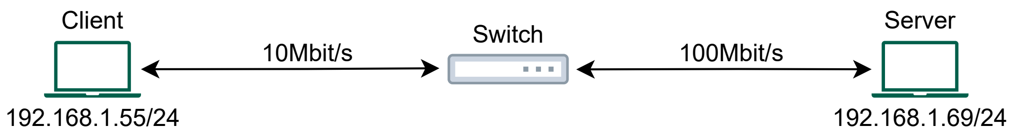


Figure 6: new testbed

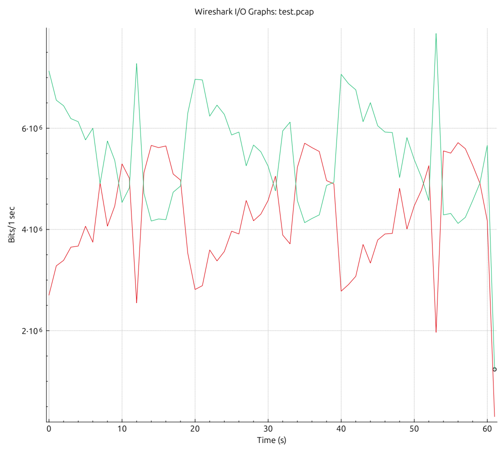
In this last experiment the packet loss probability is restored to 0%, but in addition, the channel capacity between the client and the switch has been reduced and a TCP stream has been introduced to the channel with **iperf3** tool, continuously running in background.

```
iperf3 -c 192.168.1.69 -R #from server to client (same direction as video stream)
```

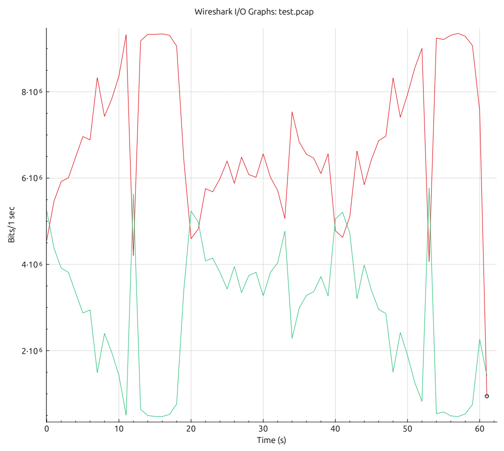
Two new video samples has been created, the first with **5 Mbits/s** encoding and the second one with **9 Mbits/s** encoding.

4.1 UDP vs TCP

In Figure 7, we have an UDP stream (the **video**) and a TCP stream (**iperf3**). As we can see, in Figure 7a, the sharing of the channel, even if not fair, it is good enough because a this bitrate (5Mbits/s), the video took half of the channel capacity on average. On the other hand, in Figure 7b, when the bitrate of the video reach almost the maximum capacity of the channel, the TCP stream is heavily influenced by the UDP one. Thus, the latter takes all the capacity it needs, compressing the TCP stream almost to zero sometimes. However, in both the cases, the VLC window is opened but only a black screen is shown, so no video at all on the screen.

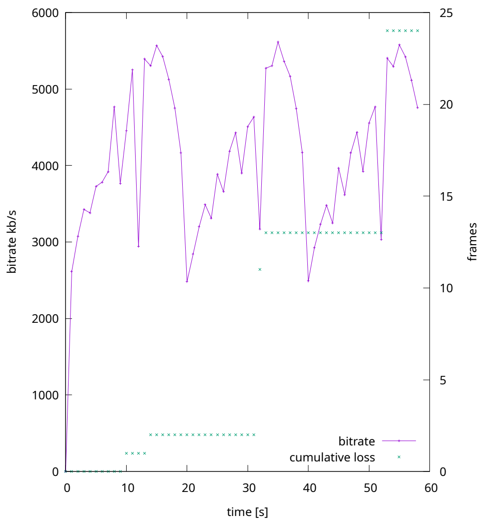


(a) bitrate 5Mbit/s

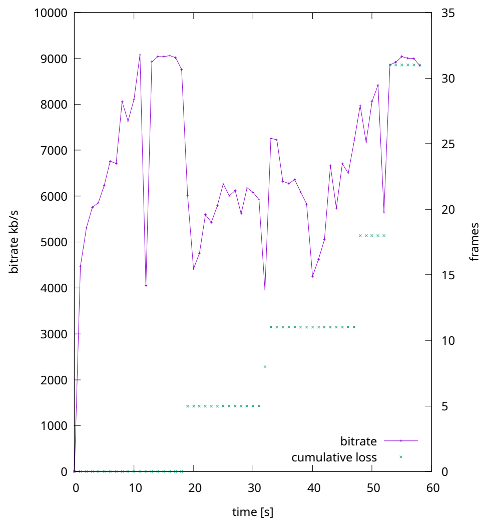


(b) bitrate 9Mbits/s

Figure 7: UDP and TCP stream



(a) bitrate 5Mbit/s



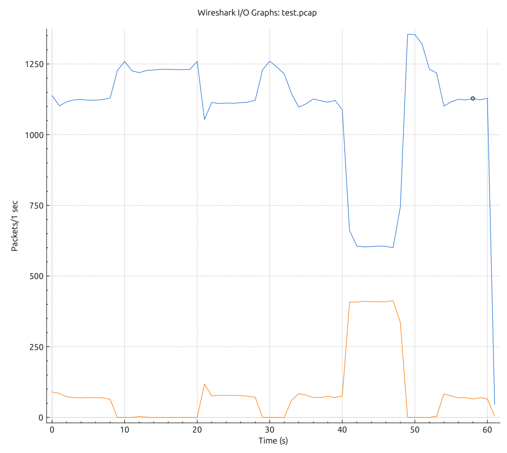
(b) bitrate 9Mbits/s

Figure 8: bitrate and cumulative loss

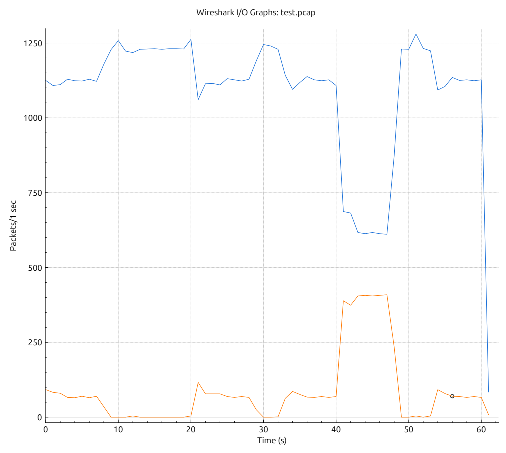
In Figure 8 it is possible to see how the video streaming bitrate changes, according to the **wireshark** plots (Figure 7). The interesting point is that here, the we registered more losses compared the first experiment, of course because of the presence of iperf3 running in background.

4.2 HTTP vs TCP

In Figure 9, we have an TCP stream (the **video**), and another TCP stream (**iperf3**). This time, due to fairness of TCP, the video streaming has room enough to run smoothly in the case of 5 Mbits/s (Figure 9a), and also in the case of 9 Mbits/s (Figure 9b). However, there is a small difference in the reproduction: in the first case the video plays smootly without lags, while in the second case, the video suffers of some micro lags (not so terrible at the end).

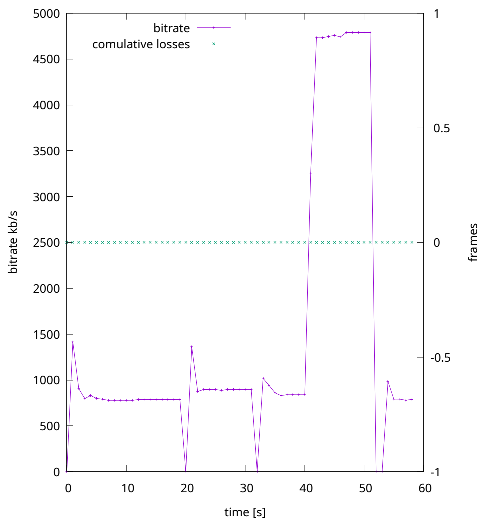


(a) bitrate 5Mbit/s

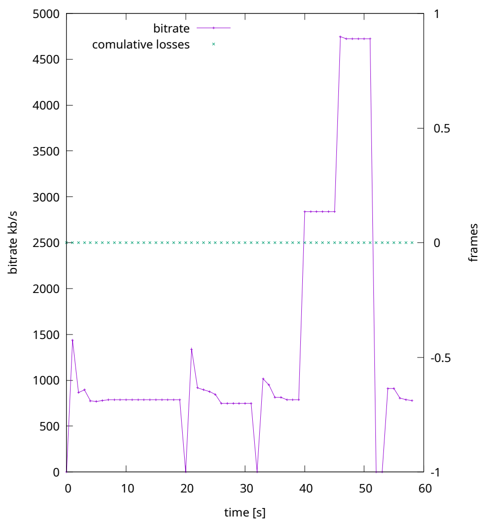


(b) bitrate 9Mbits/s

Figure 9: 2 TCP streams



(a) bitrate 5Mbit/s



(b) bitrate 9Mbits/s

Figure 10: bitrate and cumulative loss

In Figure 10 we can see how the bitrate changes according to the bandwidth taken from iperf3. The latter takes always more bandwidth because it starts before the video stream begin. At 40th second of the experiment, however, the VLC start to stream in the time between two speedtests. For this reason we have a peak indicating that video stream is predominating over the iperf3 stream, according also to Figure 9. Again no losses are registered in this situation.

In the table below it is possible to appreciate the **average throughput** registered by iperf3 while the streaming is running (Figures 7 and 9) along with the command used to perform the computation.

```
cat iperf_stats | grep receiver | awk '{total += $7; count++} END {printf "Avg speed: %.2f\n", total / count}'
```

Encoding	UDP	HTTP
5 Mbits/s	5.93 Mbits/s	8.97 Mbits/s
9 Mbits/s	3.48 Mbits/s	8.56 Mbits/s

Appendix

Script to run the experiments (only udp)

```
#!/bin/bash

i=0
echo "launch vlc server side"
ansible -m command -a "cvlc /home/$USER/Documents/report5/transcoded/sample.mp4 --sout='#udp{mux=ts,dst=192.168.1.55,port:1111}' --loop" server1
>&/dev/null &

sleep 10

echo "launch tcpdump"
sudo nohup tcpdump -i enx00e04c680059 -w test.pcap &>/dev/null &

echo "launch vlc client side"
vlc udp://:1111 --extraintf rc --rc-host=:1200 &>/dev/null &

# only when needed
echo "running iperf3 'qb' times"
./iperfbg.sh &
loop_pid=$!

echo "collecting stats ..."
rm -f stats.log
while [[ ${i} -lt $1 ]]; do
  echo "stats" | netcat 127.0.0.1 1200 >> stats.log &
  sleep 1
  i=$((i+1))
  pkill netcat
done
echo "done"

echo "killing tcpdump"
sudo pkill tcpdump
echo "killing vlc client side"
pkill vlc
echo "killing vlc server side"
ansible -m command -a "pkill vlc" server1 >&/dev/null &
echo "killing iperf3"
pkill -P $loop_pid

cat stats.log | grep "input bitrate" | tr -s ' ' | cut -d ':' -f 2 | cut -d ' ' -f 2 > bitrate
cat stats.log | grep "frames lost" | tr -s ' ' | cut -d ':' -f 2 | cut -d ' ' -f 2 > lostframes
cat stats.log | grep "discontinuities" | tr -s ' ' | cut -d ':' -f 2 | cut -d ' ' -f 2 > discontinuities

gnuplot plot.gnu
```

plot.gnu

```
set terminal wxt enhanced font ",22"
set ylabel "bitrate kb/s"
set y2label "frames"
set ytics nomirror
set y2tics nomirror
set xlabel "time [s]"
set key right bottom

plot "bitrate" using :1 title "bitrate" axis x1y1 with linespoint, "lostframes" using :1 title "cumulative loss" axis x1y2
pause 1000
```

Iperf background

```
#!/bin/bash

rm -f iperf_stats
for j in {1..10}; do
  iperf3 -c 192.168.1.69 -R >> iperf_stats
done
```

Extract and plot packet size and inter packet gap

```bash
#!/bin/bash

cat "udp/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 2 | tr -d '"' > u_time
cat "http/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 2 | tr -d '"' > h_time
cat "udp/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 6 | tr -d '"' > u_size
cat "http/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 6 | tr -d '"' > h_size
cat "udp/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 7 | tr -d '"' > u_delta
cat "http/vb6400/data.csv" | grep -v "Info" | cut -d ',' -f 7 | tr -d '"' > h_delta

paste u_time u_size u_delta > u_data
paste h_time h_size h_delta > h_data

gnuplot <<EOF &
  set terminal wxt enhanced font ",22"
  set xlabel "time[s]"
  set ylabel "packet size [Bytes]"
  set grid
  set key right
  plot "u_data" using 1:2 title "udp" with linespoint, "h_data" using 1:2 title "tcp" with linespoint
  pause 1000
EOF

gnuplot <<EOF &
  set terminal wxt enhanced font ",22"
  set xlabel "time[s]"
  set ylabel "inter packet gap [ms]"
  set grid
  plot "u_data" using 1:3 title "udp" with linespoint, "h_data" using 1:3 title "tcp" with linespoint
  pause 1000
EOF
```