IMPLEMENTATION OF A REAL-TIME VIDEO PROCESSING APPLICATION IN LITMUS-RT

By
Sai Manasa Vedantam

ABSTRACT

With the increase in demand for Video Streaming services, fueled by technological advancements, the need for Real-time Video Streaming applications for smooth and user-friendly experience is in a way inevitable. Such a Real-time Video Player that avoids video lags ensuring high quality of service & also deal with other common issues faced by using non-Real-time Video applications is being implemented using the real-time interfaces provided by Litmus-RT, with compatible versions of FFMPEG and SDL libraries to support the application, with the core being built in C language using liblitmus interface of Litmus-RT. The application is heavily tested across several Real-time schedulers along with multiple cases tweaking the real-time task parameters which are quite significant. Also, clear visualizations using tracing modules & numerical analysis tools of LitmusRT are provided to support the results.

1. INTRODUCTION

A Real-Time Video Processing application is the one which has the ability to give the users an experience that makes them feel as if they are watching things live. It is essential for more than one reason. With the current pandemic situation, the demand for video streaming services increased significantly as people are staying home most of the time & finding ways for entertainment to avoid boredom.

Also, we all know that we will be watched almost everywhere we go with the help of Surveillance cameras in shopping malls, traffic signals etc. It is observed that, in the past few years, the crime rate increased by around 30%. As almost there are no blind spots except for a very few, if the videos recorded can be processed in the right(real) time, more information can be extracted about the accused and help the authorities to take necessary actions, eventually reducing the overall crime rate. The Real-time Video Player suggested can be extended to such critical systems. Crime rate Statistics

Moreover, video streaming applications are now an integral part of any country's per capita income. With the trillions of expenses made to meet pandemic needs, almost every country is now in crisis and trying to stabilize their economy. This indicates that real-time video processing systems have good scope for growth in future. Income from Video Streaming

Such a Real-Time Video Player is built to address the above needs to a large extent, thereby providing a platform on which extensions can be built to be useful for several other relevant applications. The application is intensely tested & the results are clearly analyzed to draw conclusions that would help users to set proper values for real-time tasks.

Rest of the paper first discusses related work in section 2, then describes implementation in section 3. Section 4 describes how the system is being evaluated & presents associated results. Section 5 presents the conclusions & describes future work.

2. RELATED WORK

Video streaming applications that are currently present are good at handling various transmission and network protocols include HTTP, MMS, RTP etc; audio/video coding standards include VP8, MPEG2, MPEG4 etc. It is inappropriate to use a system that focuses on a particular transmission method, media container format and audio/video coding standard to analyze today's Internet media stream. However, these Stream analysis systems are not completely successful in handling video lags.

Based on the success of these fundamental systems, this application is designed to provide a unified solution to analyze a wide range of media data formats. To adapt to vastly different transmission protocols, media container formats, as well as audio/video coding standards, the system uses FFMPEG as its kernel while handling video lags & avoiding them. FFMPEG is an open source, powerful multimedia framework that supports multiple transmission protocols, multimedia data and provides highly efficient codec algorithm that can meet requirements of real-time audio/video analysis.

3. IMPLEMENTATION

In this section, the project structure & implementation details of the Real-time Video Player application are described. For ease of understanding, this section is further divided into sub-sections 3A to describe workflow, 3B for technologies & version information, 3C to explain implementation structure & 3D for architecture of the application respectively.

3A. WORKFLOW

The project relies on creating single threaded tasks in real-time. First, we should open the video file to work with and try to look for audio & video streams in the file using FFMPEG. Once a stream is found, we should retrieve its respective Codec information so that SDL can use that to play video on its overlay window.

Audio is handled using a queue that stores packets from the stream. A thread initiated by SDL checks this queue for audio data so that it can be decoded & played on time. Decoding & playing this raw audio may lead to noise. Hence, audio packet should be resampled before playing it to avoid noise. Video is handled by decoding a complete frame i.e the picture in the video will not be placed on the YUV display of the SDL window until each packet associated with that picture is fully decoded & ready to be used with the display.

Once the above modules are implemented, we aim to make the application work in real-time by defining the real-time job. In this case, the job is to read data & identify whether the stream is audio/video. If it is a video stream, we obtain complete picture frame by decoding all packets associated with it and update the SDL's YUV display in case of video stream. If we obtained an audio stream, we push the packet to the audio queue so that the active audio thread can deal with it. Finally, both are made to work in sync with each other & get the streaming done in real-time, thereby achieving the application goal.

3B. TOOLS & TECHNOLOGIES

The following gives a list of tools & technologies used in building the Real-time Video Player along with their version information for compatibility.

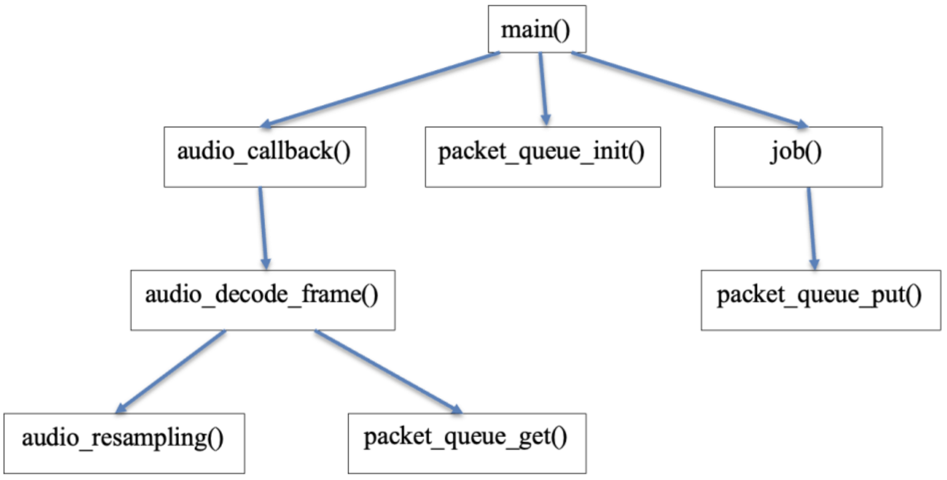
COMPONENT / LANGUAGE	TOOL	VERSION
Programming Language	C	Default to liblitmus
Kernel	Linux Kernel - LitmusRT	2016.1
Operating System	Ubuntu on Virtual machine	Default to LitmusRT
Interfaces	liblitmus for C API	2016
Libraries	FFMPEG, SDL	4.1.4, 1.2

3C. CODE STRUCTURE & MODULE INFORMATION

This sub-section describes the functions handled by each module to get the Real-Time Video Player ready for use. The sole purpose of mentioning this detail here is for ease of understanding the way modules are organized.

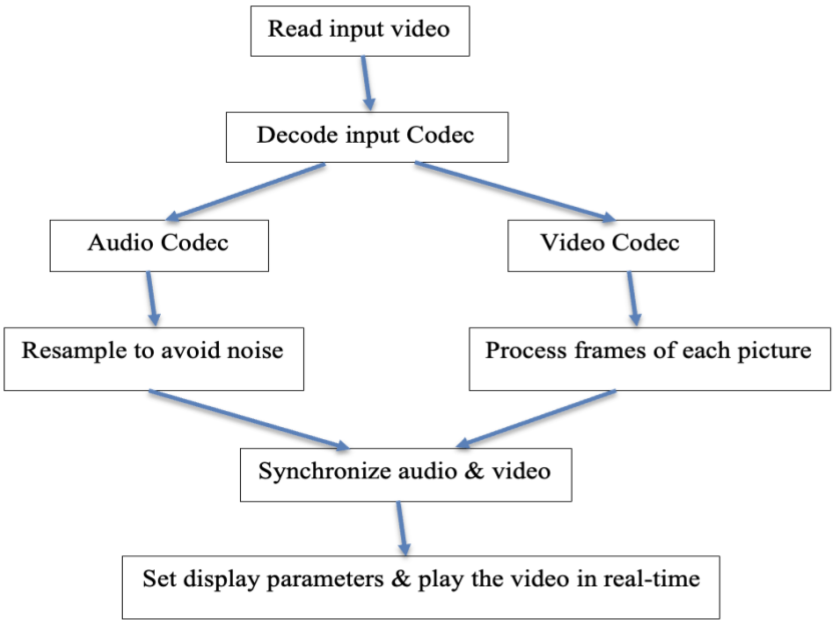
FUNCTION NAME	FUNCTIONALITY
Driver & Real-Time Job Modules	
main()	1. Sets up parameters associated with Real-Time task 2. Ensures SDL functionality & reads Input video 3. Retrieves stream information and finds audio & video streams 4. Retrieves audio context & sets desired audio specifications 5. Initializes the packet queue with audio data 6. Sets Real-Time properties & transitions from Background to Real-Time mode back and forth during task start & completion
job()	1. Decodes video frames & retrieves data in Real-Time 2. Once a frame is completely processed, set surface to display pics 3. Puts audio stream into the buffer 4. Defines the video processing job to be invoked periodically
Audio/Video Handling Modules	
audio_callback()	1. Obtains data from audio_decode_frame() & stores it in a buffer 2. Attempts to write as many bytes as the buffer size to SDL stream 3. Gets more data if we don’t have or save it for later if we have more data

audio_decode_frame()	1. If a packet is available in queue, this extracts & decodes it 2. Once we have the frame, it simply resamples 3. Copies it to the audio buffer
audio_resampling()	1. Resamples the audio data retrieved using FFMPEG before playing it & returns the size of the resampled data
Packet Handling Modules	
packet_queue_init()	1. Initializes the given packet queue
packet_queue_get()	1. Obtains the first AVPacket from the given PacketQueue
packet_queue_put()	1. Puts the given AVPacket in the audio PacketQueue



In the above diagram, all boxes which are in the same level execute one-by-one from Left to Right. Boxes which are pointed outwards from a box at higher level indicates that they will be internally called to finish execution of the calling function.

3D. ARCHITECTURE


Read input video
Decode input Codec
Audio Codec
Video Codec
Resample to avoid noise
Process frames of each picture
Synchronize audio & video
Set display parameters & play the video in real-time

4. EVALUATION

The application is being heavily tested across several built-in schedulers available in LitmusRT, along with heuristics that have theoretical as well as practical evidence for their strength. This section is further divided into sub-sections 4A to explain the metrics used, 4B to demonstrate constraints imposed & observations on the same and 4C to visualize results.

4A. METRICS USED

The metrics on which the application is tested can be divided as:

a. **Multi-Scheduler Compatibility**

LitmusRT provides built-in scheduler plugins which can be easily used for evaluating a real-time application with ease. We can use simple commands to set a specific scheduler plugin and view which scheduler is currently in use. Also, the built-in tracing tools like feather trace helps in visualizing the job releases & execution timelines to verify the way each scheduler is working with the real-time job.

List of schedulers on which the application is evaluated:

1. GSN-EDF: Global EDF scheduling
2. PSN-EDF: Partitioned, dynamic-priority EDF scheduling
3. C-EDF: Clustered EDF scheduling
4. P-FP: Partitioned, fixed-priority scheduling

Some helpful commands:

1.	To set a scheduler in LitmusRT:	setsched <Scheduler_Plugin_Name>
2.	To view the current scheduler:	showsched
3.	To obtain trace:	st-trace-schedule
4.	To draw the obtained trace:	st-draw *.bin
5.	To view the trace:	evince *.pdf
6.	To view the numerical analysis of jobs:	st-job-stats *.bin

b. **Task Parameter Analysis**

Any real-time task has three important parameters along with others that effects task execution. They are:

1. Period: Minimum length of all time intervals between release times of consecutive jobs in a Task.
2. Relative Deadline: Amount of time between job arrival time & the absolute deadline of the job.
3. Worst-case Execution time: Maximum length of time a task takes to execute on a system.

4B. CONSTRAINTS USED

There are two sets of constraints used to analyze data to obtain conclusions. In the tables below, matching color code represents mapping between constraints & observations obtained.

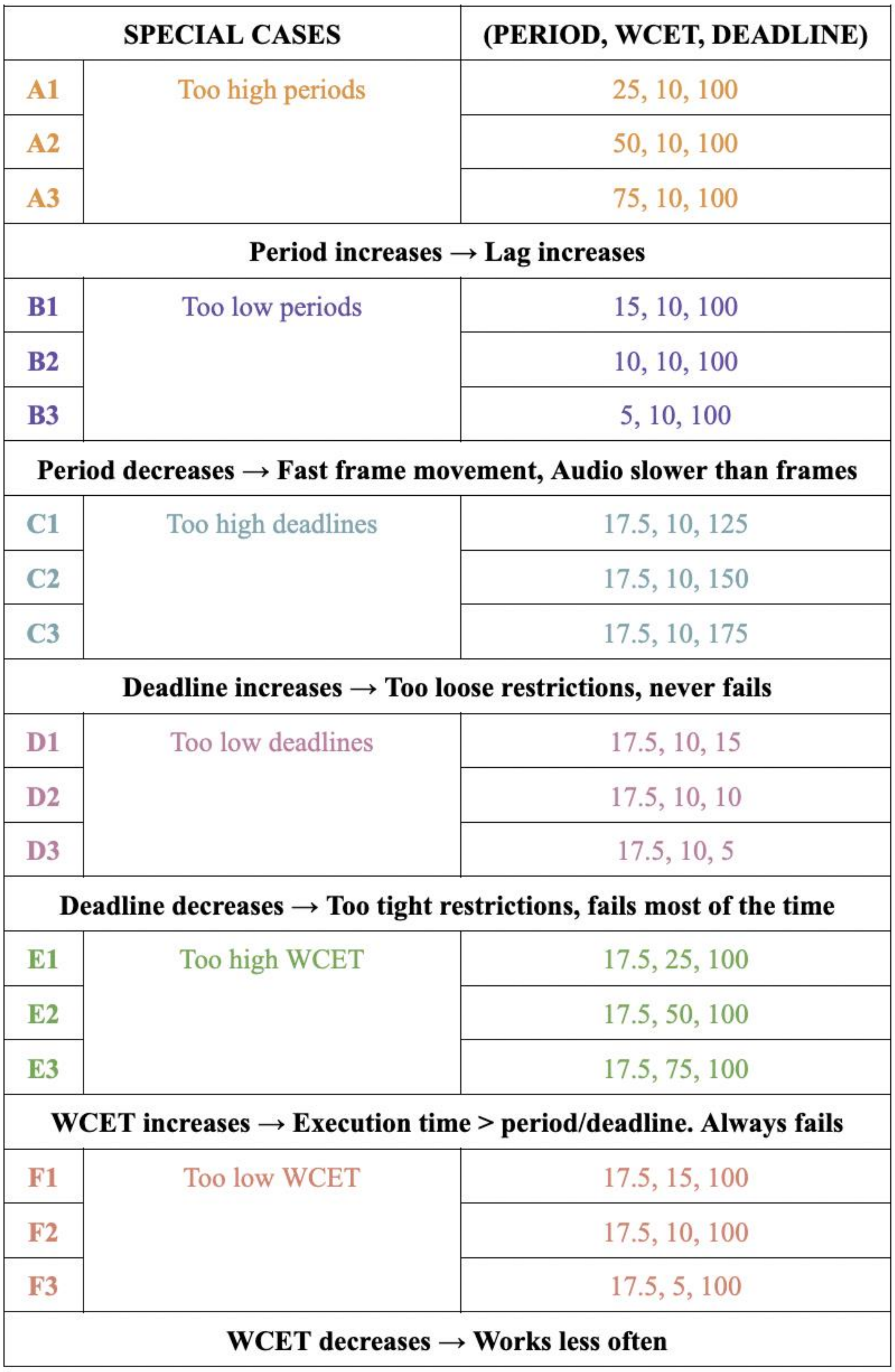
a. Constraint Set 1

CASE	CONSTRAINTS	VALUES (Period, WCET, Deadline)
1	WCET < Period & DL > Period	17.5, 10, 100
2	WCET > Period & DL > Period	17.5, 20, 100
3	WCET = Period & DL > Period	17.5, 17.5, 100
4	WCET > Period & DL < Period	17.5, 20, 15
5	WCET < Period & DL < Period	17.5, 10, 15
6	WCET = Period & DL < Period	17.5, 17.5, 15
7	WCET < Period & DL = Period	17.5, 10, 17.5
8	WCET > Period & DL = Period	17.5, 20, 17.5
9	WCET = Period & DL = Period	17.5, 17.5, 17.5
10	WCET > Period & DL > WCET	17.5, 20, 50
11	WCET < Period & DL > WCET	17.5, 15, 50
12	WCET = Period & DL > WCET	17.5, 17.5, 50
13	WCET > Period & DL < WCET	17.5, 20, 15
14	WCET < Period & DL < WCET	17.5, 15, 10
15	WCET = Period & DL < WCET	17.5, 17.5, 15
16	WCET > Period & DL = WCET	17.5, 20, 20
17	WCET < Period & DL = WCET	17.5, 15, 15
18	WCET = Period & DL = WCET	17.5, 17.5, 17.5

b. Observations obtained from Constraint Set 1

CASE	OBSERVATIONS - VIDEO			
	GSN-EDF	PSN-EDF	C-EDF	P-FP
1	Done	Done	Done	Done
2	Error	Error	Error	Error
3	Done	Done	Done	Done
4	Error	Error	Error	Error
5	Done	Error	Done	Done
6	Error	Error	Error	Error
7	Done	Done	Done	Done
8	Error	Error	Error	Error
9	Done	Error	Done	Error
10	Error	Error	Error	Error
11	Done	Done	Done	Error
12	Done	Done	Done	Error
13	Error	Error	Error	Error
14	Error	Error	Error	Error
15	Error	Error	Error	Error
16	Error	Error	Error	Error
17	Done	Done	Done	Done
18 = 9	Done	Error	Done	Error

c. **Constraint Set 2**



SPECIAL CASES		(PERIOD, WCET, DEADLINE)
A1	Too high periods	25, 10, 100
A2		50, 10, 100
A3		75, 10, 100
Period increases → Lag increases		
B1	Too low periods	15, 10, 100
B2		10, 10, 100
B3		5, 10, 100
Period decreases → Fast frame movement, Audio slower than frames		
C1	Too high deadlines	17.5, 10, 125
C2		17.5, 10, 150
C3		17.5, 10, 175
Deadline increases → Too loose restrictions, never fails		
D1	Too low deadlines	17.5, 10, 15
D2		17.5, 10, 10
D3		17.5, 10, 5
Deadline decreases → Too tight restrictions, fails most of the time		
E1	Too high WCET	17.5, 25, 100
E2		17.5, 50, 100
E3		17.5, 75, 100
WCET increases → Execution time > period/deadline. Always fails		
F1	Too low WCET	17.5, 15, 100
F2		17.5, 10, 100
F3		17.5, 5, 100
WCET decreases → Works less often		

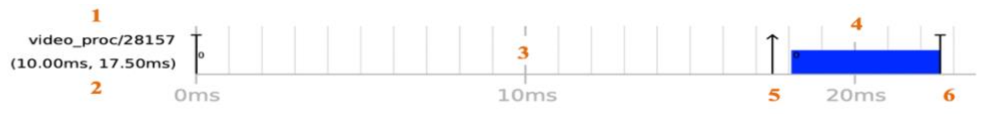
d. Observations obtained from Constraint Set 2

CASE	SPECIAL CASES - VIDEO			
	GSN-EDF	PSN-EDF	C-EDF	P-FP
A1	Done	Error	Done	Done
A2	Done	Error	Done	Done
A3	Done	Error	Done	Done
B1	Done	Error	Done	Error
B2	Done	Error	Done	Error
B3	Error	Error	Error	Error
C1	Done	Done	Done	Done
C2	Done	Done	Done	Done
C3	Done	Done	Done	Done
D1	Done	Done	Done	Done
D2	Done	Error	Done	Error
D3	Error	Error	Error	Error
E1	Error	Error	Error	Error
E2	Error	Error	Error	Error
E3	Error	Error	Error	Error
F1	Done	Done	Done	Error
F2	Done	Error	Done	Error
F3	Done	Error	Done	Error

4C. VISUALIZATION – TRACING & NUMERICAL ANALYSIS

This section deals with visualizing the results for easy interpretation. It presents both tracing as well as corresponding metrics associated with them. However, listing all the traces related to each of the above cases may cause clutter & hence, a reference to them is provided below.

Interpretation of scheduling real-time jobs on a specific scheduler:



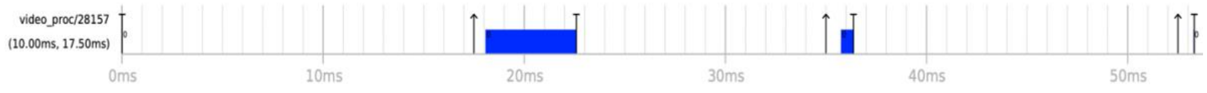
1. Application Name / Process ID
2. (Worst Case Execution Time, Period)
3. Timeline
4. Job Execution
5. Job Release Marker
6. Job Completion Marker

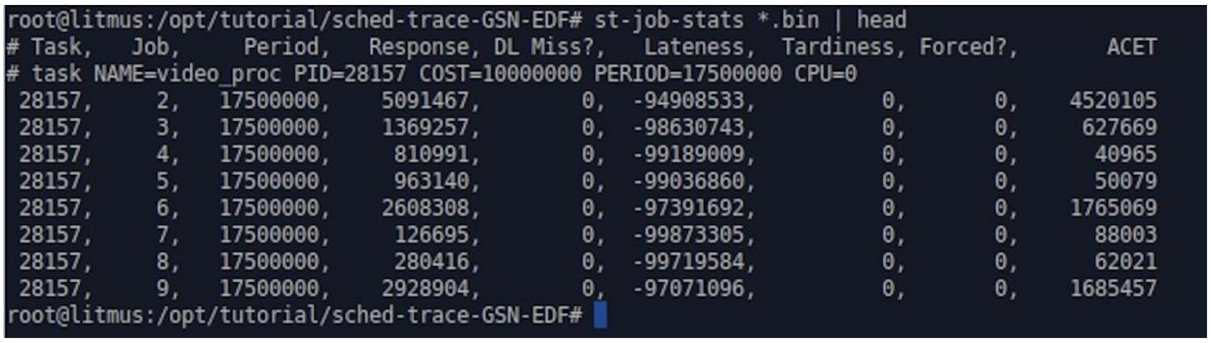
In the numerical analysis, we visualize job statistics which include important factors like:
1. Task Number
2. Deadline Missed?
3. Average Case Execution Time
4. Period
5. Response Time etc.

For a chosen scheduler, jobs will be scheduled as per its scheduling policy. If the jobs cannot be scheduled in such a manner, the application leads to erroneous situation which implies that the task cannot be executed in real-time with the parameters assigned to it. In that case, parameter tuning is essential to get the system work in proper condition and deliver expected results. If all jobs of a task can be scheduled under a scheduling algorithm, the task can be considered successful. When we try to trace the job execution, if the Task system can be scheduled, we get its information. If not, we get bad data that won't help us fetch any useful results. Hence, there won't be any trace or statistical analyses for Error cases.

Case 1 – Result Snapshot

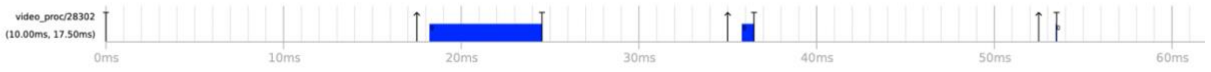
a. GSN-EDF

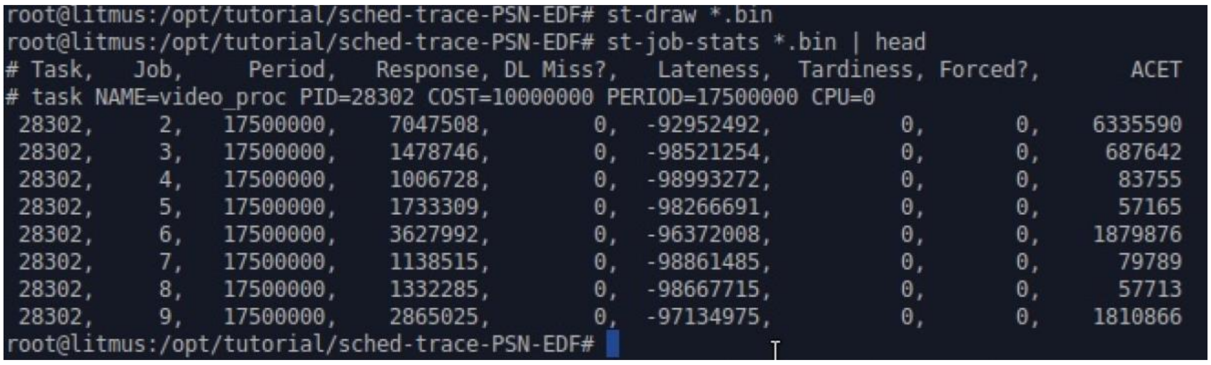




```
root@litmus:/opt/tutorial/sched-trace-GSN-EDF# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=28157 COST=10000000 PERIOD=17500000 CPU=0
 28157,     2,   17500000,    5091467,        0,  -94908533,          0,       0,    4520105
 28157,     3,   17500000,    1369257,        0,  -98630743,          0,       0,     627669
 28157,     4,   17500000,     810991,        0,  -99189009,          0,       0,      40965
 28157,     5,   17500000,     963140,        0,  -99036860,          0,       0,      50079
 28157,     6,   17500000,    2608308,        0,  -97391692,          0,       0,    1765069
 28157,     7,   17500000,     126695,        0,  -99873305,          0,       0,      88003
 28157,     8,   17500000,     280416,        0,  -99719584,          0,       0,      62021
 28157,     9,   17500000,    2928904,        0,  -97071096,          0,       0,    1685457
root@litmus:/opt/tutorial/sched-trace-GSN-EDF#
```

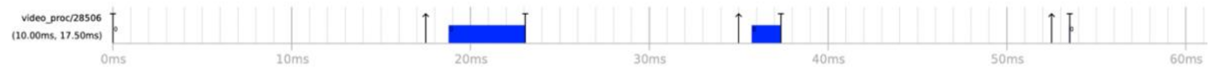
b. PSN-EDF

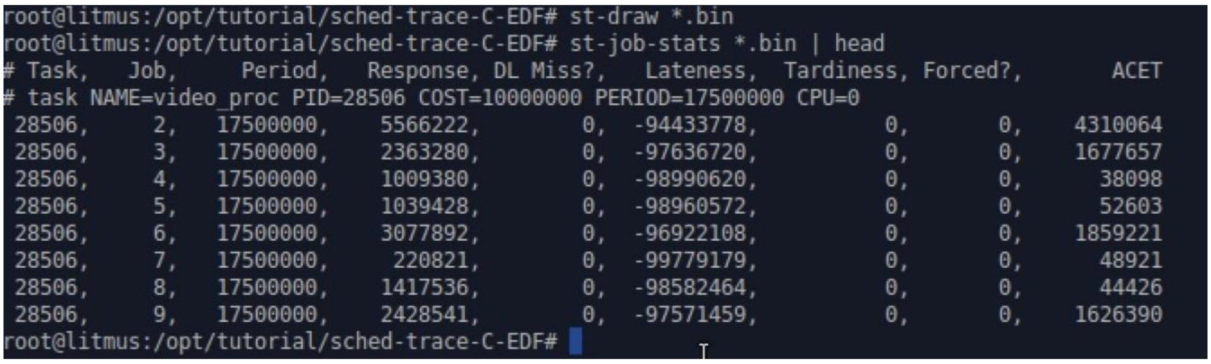




```
root@litmus:/opt/tutorial/sched-trace-PSN-EDF# st-draw *.bin
root@litmus:/opt/tutorial/sched-trace-PSN-EDF# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=28302 COST=10000000 PERIOD=17500000 CPU=0
 28302,     2,   17500000,    7047508,        0,  -92952492,          0,       0,    6335590
 28302,     3,   17500000,    1478746,        0,  -98521254,          0,       0,     687642
 28302,     4,   17500000,    1006728,        0,  -98993272,          0,       0,      83755
 28302,     5,   17500000,    1733309,        0,  -98266691,          0,       0,      57165
 28302,     6,   17500000,    3627992,        0,  -96372008,          0,       0,    1879876
 28302,     7,   17500000,    1138515,        0,  -98861485,          0,       0,      79789
 28302,     8,   17500000,    1332285,        0,  -98667715,          0,       0,      57713
 28302,     9,   17500000,    2865025,        0,  -97134975,          0,       0,    1810866
root@litmus:/opt/tutorial/sched-trace-PSN-EDF#
```

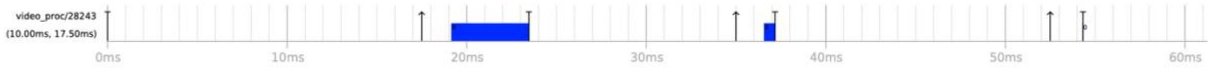
c. C-EDF

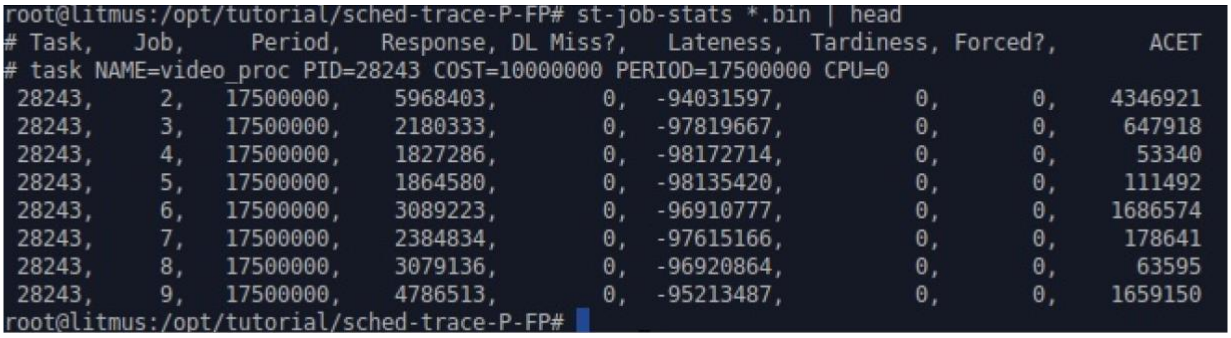




```
root@litmus:/opt/tutorial/sched-trace-C-EDF# st-draw *.bin
root@litmus:/opt/tutorial/sched-trace-C-EDF# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=28506 COST=10000000 PERIOD=17500000 CPU=0
 28506,     2,   17500000,    5566222,        0,  -94433778,          0,       0,    4310064
 28506,     3,   17500000,    2363280,        0,  -97636720,          0,       0,    1677657
 28506,     4,   17500000,    1009380,        0,  -98990620,          0,       0,      38098
 28506,     5,   17500000,    1039428,        0,  -98960572,          0,       0,      52603
 28506,     6,   17500000,    3077892,        0,  -96922108,          0,       0,    1859221
 28506,     7,   17500000,     220821,        0,  -99779179,          0,       0,      48921
 28506,     8,   17500000,    1417536,        0,  -98582464,          0,       0,      44426
 28506,     9,   17500000,    2428541,        0,  -97571459,          0,       0,    1626390
root@litmus:/opt/tutorial/sched-trace-C-EDF#
```

d. P-FP

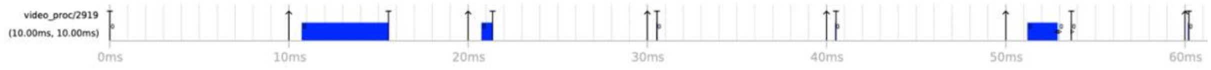


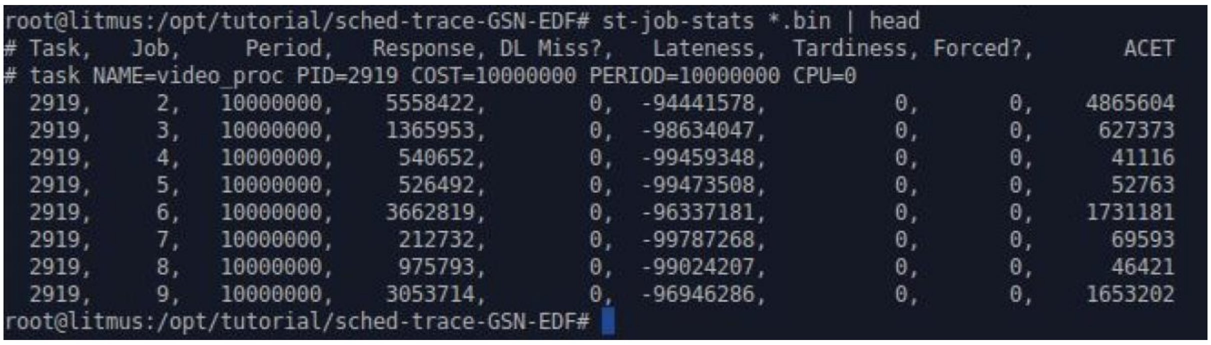


```
root@litmus:/opt/tutorial/sched-trace-P-FP# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=28243 COST=10000000 PERIOD=17500000 CPU=0
 28243,     2,   17500000,    5968403,        0,  -94031597,          0,       0,    4346921
 28243,     3,   17500000,    2180333,        0,  -97819667,          0,       0,     647918
 28243,     4,   17500000,    1827286,        0,  -98172714,          0,       0,      53340
 28243,     5,   17500000,    1864580,        0,  -98135420,          0,       0,     111492
 28243,     6,   17500000,    3089223,        0,  -96910777,          0,       0,    1686574
 28243,     7,   17500000,    2384834,        0,  -97615166,          0,       0,     178641
 28243,     8,   17500000,    3079136,        0,  -96920864,          0,       0,      63595
 28243,     9,   17500000,    4786513,        0,  -95213487,          0,       0,    1659150
root@litmus:/opt/tutorial/sched-trace-P-FP#
```

Case B2 – Result Snapshot

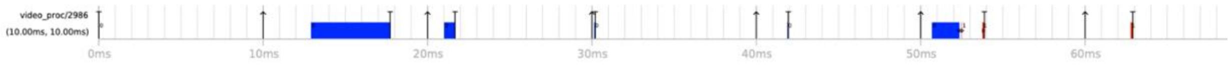
a. GSN-EDF

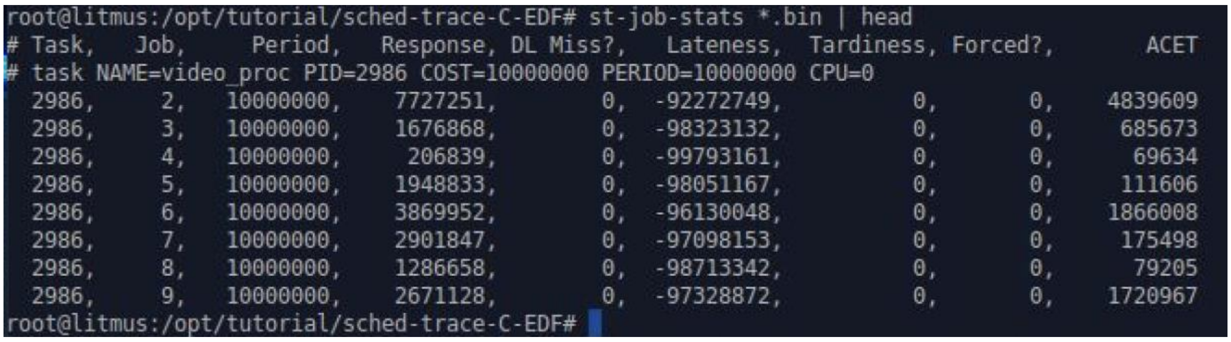




```
root@litmus:/opt/tutorial/sched-trace-GSN-EDF# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=2919 COST=10000000 PERIOD=10000000 CPU=0
  2919,     2,   10000000,    5558422,        0,  -94441578,          0,        0,    4865604
  2919,     3,   10000000,    1365953,        0,  -98634047,          0,        0,     627373
  2919,     4,   10000000,     540652,        0,  -99459348,          0,        0,      41116
  2919,     5,   10000000,     526492,        0,  -99473508,          0,        0,      52763
  2919,     6,   10000000,    3662819,        0,  -96337181,          0,        0,    1731181
  2919,     7,   10000000,     212732,        0,  -99787268,          0,        0,      69593
  2919,     8,   10000000,     975793,        0,  -99024207,          0,        0,      46421
  2919,     9,   10000000,    3053714,        0,  -96946286,          0,        0,    1653202
root@litmus:/opt/tutorial/sched-trace-GSN-EDF#
```

b. C-EDF





```
root@litmus:/opt/tutorial/sched-trace-C-EDF# st-job-stats *.bin | head
# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardiness, Forced?,       ACET
# task NAME=video_proc PID=2986 COST=10000000 PERIOD=10000000 CPU=0
  2986,     2,   10000000,    7727251,        0,  -92272749,          0,        0,    4839609
  2986,     3,   10000000,    1676868,        0,  -98323132,          0,        0,     685673
  2986,     4,   10000000,     206839,        0,  -99793161,          0,        0,      69634
  2986,     5,   10000000,    1948833,        0,  -98051167,          0,        0,     111606
  2986,     6,   10000000,    3869952,        0,  -96130048,          0,        0,    1866008
  2986,     7,   10000000,    2901847,        0,  -97098153,          0,        0,     175498
  2986,     8,   10000000,    1286658,        0,  -98713342,          0,        0,      79205
  2986,     9,   10000000,    2671128,        0,  -97328872,          0,        0,    1720967
root@litmus:/opt/tutorial/sched-trace-C-EDF#
```

Results associated with rest of the cases can be viewed Here on my Github.

5. CONCLUSIONS AND FUTURE WORK

The implementation of a Real-Time Video Processing application based on LitmusRT, built using FFMPEG and SDL is clearly described in this paper. This application supports several audio/video coding standards & formats, several transmission protocols etc. along with satisfying real-time requirements & avoiding lags. From the evaluation, it can be concluded that the performance of the application depends heavily on Task parameters and hence, they should be chosen very carefully. For instance, too large periods cause lags while too small periods lead to faster frame movement which is out of sync with the audio. Similarly, if the Worst-case execution time is greater than period/deadline, task tends to fail always. As such, deadlines have the capacity to pose or relax restrictions on the application. Also, choosing a proper scheduler helps to finish job in right time with minimum response time & avoids lags. As an extension, this application can be linked with heavy data generating applications to get more benefit out of processing them in real-time.

REFERENCES

[1] X. Lei, X. Jiang and C. Wang, "Design and Implementation of a Real-Time Video Stream Analysis System Based on FFMPEG," 2013 Fourth World Congress on Software Engineering, 2013, pp. 212-216, doi: 10.1109/WCSE.2013.38

[2] T. Chantem, X. Wang, M. D. Lemmon and X. S. Hu, "Period and Deadline Selection for Schedulability in Real-Time Systems," *2008 Euromicro Conference on Real-Time Systems*, 2008, pp. 168-177, doi: 10.1109/ECRTS.2008.35.
[3] https://linuxfromscratch.org/blfs/view/svn/multimedia/sdl.html
[4] https://www.litmus-rt.org/tutor16/manual.pdf
[5] http://dranger.com/ffmpeg/tutorial01.html