Localized Application for Video Capture for a Multimedia Sensor Node with Name-Based Segment Streaming

by

Zarah Khan

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Martin Reisslein, Chair
Adolph Seema
Antonia Papandreou-Suppappola

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

The Internet of Things (IoT) has become a more pervasive part of everyday life. IoT networks such as wireless sensor networks, depend greatly on the limiting unnecessary power consumption. As such, providing low-power, adaptable software can greatly improve network design. For streaming live video content, Wireless Video Sensor Network Platform compatible Dynamic Adaptive Streaming over HTTP (WVSNP-DASH) aims to revolutionize wireless segmented video streaming by providing a low-power, adaptable framework to compete with modern DASH players such as Moving Picture Experts Group (MPEG-DASH) and Apple's Hypertext Transfer Protocol (HTTP) Live Streaming (HLS). Each segment is independently playable, and does not depend on a manifest file, resulting in greatly improved power performance. My work was to show that WVSNP-DASH is capable of further power savings at the level of the wireless sensor node itself if a native capture program is implemented at the camera sensor node. I created a native capture program in the C language that fulfills the name-based segmentation requirements of WVSNP-DASH. I present this program with intent to measure its power consumption on a hardware test-bed in future. To my knowledge, this is the first program to generate WVSNP-DASH playable video segments. The results show that our program could be utilized by WVSNP-DASH, but there are issues with the efficiency, so provided are an additional outline for further improvements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

As an ever-growing popular topic, wireless sensor networks have become an integral part of society with the development and implementation of the Internet of Things (IoT). These networks are expected to regularly gather and send desired information from a variety of sensors. In addition, they must do so while also efficiently utilizing power and memory of the sensors, as well as delivering the information in a timely and reliable manner. The data collected can is then utilized for a variety of applications [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. Certainly, with the rising demand for reliable video streaming for purposes such a surveillance, there is a clear demand to establish a reliable low-power network running in real-time [14].

In particular, the need for low power usage stems from the condition of the sensors. Often, these sensors are placed in remote environments far from the recipient of the data. These sensors are limited by hardware constraints and limited resource access such as a reliable power supply. Often, these sensors work on a limited battery and are only capable of a certain level of computing power [15], [16], [17], [18], [19], [20]. The limited computing and storage resources of typical wireless sensor nodes [21], [22], [23] have spurred the developed of specialized video coding mechanisms, e.g., [24], [25], [26], [27], [28], [29], [30].

Furthermore, it can be inefficient, inconvenient, or even dangerous to manually replace such power supplies too often. Improving the hardware itself can have improvements, but they are often limited. Frequently replacing an old sensor with a newer one with a more efficient processor, for example, is an inefficient and costly solution. It would not compensate greatly for inefficient, poorly-designed networks or software. Hardware can only make up for poorly implemented software or inefficiently designed networks to a limited degree, so many developers choose to improve software as a more cost-effective measure. Or they prefer to look at the overall design. It is far better to improve the overall network design and framework to improve the lifespan of sensors, while understanding the trade-offs between extending lifespan and meeting the desired application requirements [19].

1.2 Previous Work

Seema et. al. [1] looked to improve video-streaming from these sensor nodes under such power constraints. Their approach uses the Wireless Video Sensor Network Platform compatible Dynamic Adaptive Streaming over HTTP (WVSNP-DASH) framework to capture and send name-based, video segments [2]. This framework takes a streamlined approach to conventional streaming frameworks such as Moving Picture Experts Group - Dynamic Adaptive Streaming over HTTP (DASH or MPEG-DASH) [31], [32]. WVSNP-DASH utilizes a “segment naming syntax,” which simply means the metadata is in the name of the video segment itself, to remove the need to depend on an external manifest file, as current models do [1]. Simply put, these segments can be hosted on a server and based on their names the WVSNP-DASH Player (WDP) can fetch them in the correct order. Because each of these segments contains the basic metadata (such as the quality or order in the sequence) in the name, each segment is essentially a self-contained video file that may be played independently from other files. These segments are labelled by their sequence number, the playback mode (whether LIVE or Video on Demand (VOD)), the total number of segments in the sequence, and the video container format [2].

For their contribution, Seema et al. [1] were able to establish a benchmark for measuring power consumption at a wireless sensor node when using manifest file dependent streaming (like HLS) vs. segmented name based streaming (like WVSNP-DASH). First, they provide a power profile of both frameworks WVSNP-DASH and HLS. The power profile includes the power consumed for all the typical steps of retrieving video over the internet after a request has been made for the video: capturing the video segments, storing them, transmitting them over the wireless network, and playing back the video segments using the player open on the system of the individual who requested the video. They then provide extensively-detailed measurements of the current draw at the wireless sensor node for both streaming frameworks.

They next profile design choices that could further influence power consumption. For example, there was consideration for the choice in video library: FFmpeg [33] vs. GStreamer [34]. They also looked at data movement over a USB vs. a Camera Serial Interface (CSI). And finally, they looked at the power profile when using hardware acceleration versus running the software only when encoding.

In their experiment, they profiled the power consumption of the name-based WVSNP-DASH format against Apple's manifest-file dependent HTTP Live Streaming (HLS) format. Their power measurement setup included several components. First, a remote client was setup that hosts the video playback that sends the request for the video. The client used the WVSNP-DASH Player (WDP) [2] and HLS was streamed using JWPlayer 6 with HLS-supported plug-in, with version 32 of Google Chrome serving as the web browser [1]. Next, the wireless sensor node was constructed using a *NXP i.MX6 ARM Cortex-A9, 1.2 GHz Quad core 2 GB node development board* and had attached to it both a USB webcam and a CSI attached *Wandcam* [35]. Hosting on a local server, measurements were taken both over WiFi and over Ethernet. A current clamp was used to measure the current draw in milliamperes (mA) and voltage (V) was measured in volts using an oscilloscope [1]. It is important to understand the setup done in their work as in Chapter 5 we will discuss how we intend to very closely recreate the experiment using our own contribution.

The results of the experiment showed that when comparing WVSNP-DASH and HLS for miniaturized wireless sensor nodes, WVSNP-DASH generally performed with greater reduced power consumption over HLS. However, it was noted that there was the potential for additional power saving should a local program be implemented, as their experiment relied on an inefficient script [1]. This Linux script would simply repeatedly open and close the open source video library FFmpeg, which was used to run the video capture, and saves each segment in the WVSNP-DASH format. This opening and closing of the program led to unnecessary power loss that could be saved if FFmpeg was only invoked once, segments are encoded, then have ffmpeg close once the video segments are collected. The primary conclusion to take away from their work is that if a native capture program were utilized instead of the inefficient script used in their work, the power saving between WVSNP-DASH and HLS would be even greater, as WVSNP-DASH has already shown to have a reduced power consumption. [1] For our own contribution, we intend to provide a native capture program to recreate this experiment.

1.3 Contribution

The primary contribution of this work is to provide and outline the means to further improve upon the work done by Seema et. al., as there were several limitations in their experimental setup [1].

In our work, we implement a modular standalone C application to capture LIVE video that is WVSNP-DASH compatible, based on the need stated in Chapter 1.2 [1]. Our native capture program utilizes the open source FFmpeg library tools, which provide the necessary support for capturing and encoding the video [33]. Once the program was written to suit our needs, we analyzed the efficiency of the video capture between our native capture program and the script. This is further explained in Chapter 4.1. In addition, we measured the memory usage and time-lapse, as both are additionally good indicators of a program's efficiency. We chose to look at the efficiency of just the program and the script themselves, not utilizing any framework, to ensure our program was suitably optimized before recreating the previous experiment [1]. Our method and results are discussed in Chapter 3. To the author's knowledge, there does not exist a program like this so there is no means of comparison. Thus, we needed to make our own goals for the program and establish a rigorous, consistent test to show enough improvement to allow the program to move onto the next phase of testing. While we have optimized our program to testing capacity, we also acknowledge that further optimization is possible. Due to time constraints, we were only able to achieve a certain level of utility from the program, so we will outline how to improve the program's efficiency. The detailed description of our program and suggestions for further optimization are included in Chapter 4.

Afterward, we outline the future work to be done with the native capture program and we provide the experimental setup to test our native capture program against HLS and the previous script over WVSNP-DASH similar to the previous work [1]. We outline in Chapter 5 our suggestion for the experimental setup to be conducted on similar hardware. For future testing, we have already taken the liberty of setting up the WDP and two additional players, including HLS, so that we may soon begin the hardware-based measurement. Details of how to conduct offline setup of these players are also included in Chapter 5.

To summarize, the rest of this paper is organized as follows: Chapter 2 provides the background information of video streaming and power profiling. Chapter 3 provides the methodology of the power profiling, the measurements collected, and a discussion of the results. Chapter 4 provides a more detailed explanation of our native capture program and suggestions for further optimization based on our findings. Chapter 5 outlines the future work to be done, and outlines how to recreate the experiment of the previous work. Finally, we include references and our native capture program in the appendix.

CHAPTER 2

BACKGROUND

2.1 Video Streaming Using DASH

The Moving Picture Experts Group (MPEG), a primary authority for video and audio quality standards, released the Dynamic Adaptive Streaming over HTTP (DASH) technique with the intention to provide quality video streaming frameworks alongside the rise in the usage of smartphones and other multimedia devices [36]. The intention was to replace the outdated mindset of the Real-Time Transfer Protocol (RTP), as Sodagar [31] shows several flaws to the system. While RTP worked fine in small, managed network settings, relied too heavily on session-management to deliver audio and video. It also would be regularly be locked by firewalls, something that is not an issue with HTTP. Furthermore, Sodagar points out that managed networks are on their way out as content delivery networks (CDN), which often do not have RTP-support, are becoming the norm. CDN's are essentially a decentralized network prioritizing high-quality content delivery. The infrastructure of the internet has been made to well support HTTP and this approach to content delivery [31].

The reason for streaming to develop in this direction is that with video streaming being some of the most common content delivered through a network, it is critical to be able to do so reliably. Adaptive streaming is more suited to the dynamic nature of IP networks compared to streaming platforms before it, as a high amount of bandwidth is required for higher quality audio and video streams, but such bandwidth is not always available [37]. Ideally, we want to prioritize having a reliable stream, so if the network is less reliable, and high bandwidth is not guaranteed, we want to see our network dynamically adapt by lowering the video or audio quality to maintain the overall Quality of Experience (QoE).

DASH streaming has two components. The first is a manifest file that contains all the relevant metadata for the client to be able to fetch and play segments in the correct sequence. The second part are media segments that determine the format of the content into such formats as the MP4 format [37]. MPEG-DASH and HLS are both streaming formats based on the DASH standard. Both utilized a manifest file, although each use their own format of manifest file and segment formats. This means that hypothetically, for two devices to share content with each other, one device must support the other's client protocol [31]. Both formats work by taking some multimedia content, then encoding it, and then dividing it into multiple segments. The first segment is usually the initialization segment, which contains the required information to initialize the DASH client's media decoder [31]. This is the most important segment as the entire stream of segments will not function properly if this initialization segment is not processed. The DASH client then uses the information to request the segments. For live streaming, DASH has been found to have issues with end-to-end delay, in that, there is always a certain degree of waiting for the next segment [37].

HLS is the DASH standard developed by Apple. It is similar to the MPEG-DASH standard, as stated above, but it utilizes its own manifest file format and media segment extension formats. In contrast, MPEG-DASH is codec-agnostic. Being agnostic, this standard can support any codec format used by the encoded content, and can support said content whether it is multiplexed or unmultiplexed encoded content [31]. This makes MPEG-DASH more flexible compared to the HLS standard. For example, MPEG-DASH does not require a browser to directly support DASH in the video element. HLS on the other hand, requires that browsers using the standard to be written in such a way to properly support its manifest files [2]. However, because both utilizes a manifest file for the metadata and an initialization segment to begin the stream, this makes the segments heavily dependent on each other. During video playback, these dependencies and generation tools must be maintained by the video server and the client. HLS was also the standard compared against WVSNP-DASH in the work by Seema et al. [1], and will primarily be the focus of our work. Apple HLS is an established, commercial product, and the goal of their work and our own is to see if WVSNP-DASH can outperform HLS.

2.2 WVSNP-DASH

Seema et al. [1], [2] have developed a solution to address the issue of previous adaptive segmented video streaming frameworks, especially when streaming from small, wireless, multimedia sensor nodes. Rather than having segmented video with dependencies held by a manifest file, this unique approach suggests that each video segment be independently playable by having the essential metadata held within its name. While this requires that the files adhere to a rigid name syntax, shown below, it essentially removes the need for a manifest file, or initialization in order to retrieve and playback the desired segmented video from the sensor network.

The naming syntax was inspired by the Backus-Naur Form: <*filename*>-<*maxpresentation*>-<*presentation*>-<*mode*>-<*maxindex*>-<*index*>.<*ext*> -- where <*filename*> is the stream's unique identifier, <*maxpresentation*> is an integer value of the maximum possible stream quality, <*presentation*> in contrast is the actual stream quality value (with 0 being the lowest quality possible), <*mode*> simply determines if the stream is live playback (LIVE) or video on demand (VoD), <*maxindex*> indicates the number of segments total request, <*index*> indicates a segment's position in the sequence from 1 to the <*maxindex*>, and <*ext*> is the video container format (e.g. *.mp4*) [2], [38]. In our program, we followed this guideline when generating the filenames of the video segments in order to make the native capture program adhere to the rigid naming format. This information is sufficient to remove the need for a manifest file [2]. This removes the need to have the playback conducted in the right order as MPEG-DASH and HLS require.

WVSNP-DASH solves a continually growing issue of meeting modern demand for high quality video. Video streams are some of the most commonly shared and distributed content on the internet. There's a particular demand for high-quality, reliable streams of VoD and LIVE video. To prevent congestion from a large video file size, the most video nowadays are segmented over the Internet Protocol (IP) communication network. This allows for adaptive streaming of the video quality and bitrate between segments. This adaptability makes it possible to maximize the performance of video streaming given certain conditions or trade-offs [39], [40].

Currently, the majority of modern web browsers support video playback under HTML5 [37], [41], [42], [43]. While HTML5 allows for an easy implementation of video players by use of the video tag, the challenge comes from the diverse formats and requirements of different media formats, often times requiring workarounds to properly play. Furthermore, if the bitrate or the quality is changed, it can require re-downloading the entire video file again [2]. WVSNP-DASH utilizes its own player, the WDP, for streaming video with the HTML5 element. The player follows the specification for DASH.

WDP is similar to MPEG-DASH in that both do not require a browser to directly support DASH like HLS does. However, WDP does not require a server to format the multimedia data [2]. Beyond TCP/IP networks, WVSNP-DASH also supports non-traditional protocol networks such as Zigbee, which are not supported by existing DASH players [2]. There is also an additional benefit from name-based video segmentation in regards to backwards-compatibility. The flexibility of WDP comes from the fact that it is essentially container agnostic. Therefore, because WVSNP-DASH, like MPEG-DASH, is codec agnostic, will work with most web browsers (e.g., Chrome, Windows Internet Explorer, Firefox), support HTML5, and work with non-TCP/IP networks, WVSNP-DASH is a solid choice for interfacing with wireless sensor networks that conduct video capture because of it does not require any redesign of video containers or file formats [2].

WVSNP-DASH still needs to be improved in a few key areas. Currently, the most recent measurements of the performance of WVSNP-DASH were not measured to reflect the nature of dynamic streaming. We would expect bitrates, the pixel format, or frame rates to fluctuate due to the nature of wireless networks and video playback. In the work conducted by Seema et al., [1] all three variables were measured as preset constants. While the WDP can switch between both BIG and SMALL in real time to provide a degree of dynamic adaptability, this method is rather limited. In our own program, these three variables are also fixed so that our program could be used to recreate the same test procedure. In future, this will need to be taken into account.

2.3 Power Testing

IoT networks often utilize miniature devices such as wireless video sensors. There is an ever-growing urgency to look at the power consumption of these hardware-limited devices, as these devices have continually developed more complex and more ubiquitous multimedia applications [13], [44], [45], [46]. These more complex devices require careful understanding of how power is managed. Many times, devices such as wireless sensors are placed in remote areas, and likely depend on battery power. Sensors in these conditions have to settle for certain tradeoffs. In regards to size, the devices must be self-contained and self-reliant, containing the necessary sensors, power source, and CPU. But they must do so while also being as small as possible for convenience. These devices must also have the CPU power for their desired tasks, while also taking into account power efficiency, as they will likely run on a battery. Ideally, these sensor devices would balance the need to maximize efficiency and minimize power consumption, while also meeting desired performance goals set by the developer.

For sensor network design, power supply and consumption are arguably the most important factor to consider. There have been several studies that show how power affects the operation of a sensor network [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57]. There have only been a few studies to focus on the power consumption at the sensor node level [58], [59], [60]. These can be the most important as these nodes can have the most limited resources. In this paper, we are interested on understanding and measuring the power consumption between a name-based video segmentation stream [1] and the current manifest-based video segmentation stream method, as there is evidence that the former can save power in wireless sensor networks.

For our native capture program, we wanted to make sure that it was suitably optimized to participate in the previous experiment [1]. To do so, we conducted software-based testing to measure improvements on power consumption. Our goal was to properly optimize the native capture program that there will be considerable improvements when the previous experiment is recreated. We believed that both the script and our native capture program ought to be tested in an isolated environment to better understand the contribution to the power consumption of both pieces of software. We determined that software testing would be best suited for our purpose as changes can be made quickly to the program and then observed on the computer.

With that goal in mind, we studied how to best conduct software-based power testing. We were interested in studying a software's power draw, which can be measured in Watts, or the product of a voltage V in volts and a current I in amperes. We analyze power over a time t in seconds. There are several existing tools such as the Linux tools *PowerTop* [1] and *powerstat* [61]. *Powerstat* is the software we chose to use in our power measurements and is further discussed in Chapter 3. There are also process-specific measurement software libraries, such as *PowerAPI* and kernel-specific libraries such as *powerscripts* and *powerman* [1]. Software-based testing such as this can help establish a general trend or provide a quick means to determine if any improvements to software result in improved power efficiency. It also has the benefit of quick and easy implementation, as it allows our team to make quick changes to the native capture program and then shortly test the performance after. However, it is worth noting that running any additional software, ironically including measurement software, affects the measurement of power consumption, as the measurement software also requires power itself. Specifically in-system methods of testing affect the device under test (DUT) because they require power to function [1]. Limitations to of our work will be further explored in subsequent chapters.

As useful as software-based testing is, we can further get better results through hardware testing. Testing with hardware allows the power measurement devices to be separate from the device being tested and isolates the system for a better, more accurate measurement. There are several different approaches for to measuring the energy consumption of wireless devices [1]. Of the different methods, the one of particular interest is the inductor method, popularly used for heavy engineering tools, as it was used in previous work [1]. Essentially, current is measured by sampling the voltage at the clamp inductor by the electric field from the wire that supplies the load current. While this method allows for high sampling rates, it also requires frequent recalibration [1].

The power measurement setup shown in previous work [1] follows the inductor power measurement method, and it will be the setup of future testing with our own native capture program. Figure 2.1 below shows the setup we will intend to recreate in future work. To follow previous work, the following setup will be created. The board used would be a NXP i.MX6 ARM Cortex-A9, 1.2 GHz Quad core, 2 GB node development board, or a very similar board. To measure the current, we would use a 10 μA resolution current clamp attached to the power wire of the board. A digital oscilloscope with 100 MHz bandwidth, a sampling rate of 1 gigasamples/s, and 12-bit enhanced resolution would be attached to the 5V jack on the board to measure voltage. While alternatives exist such as

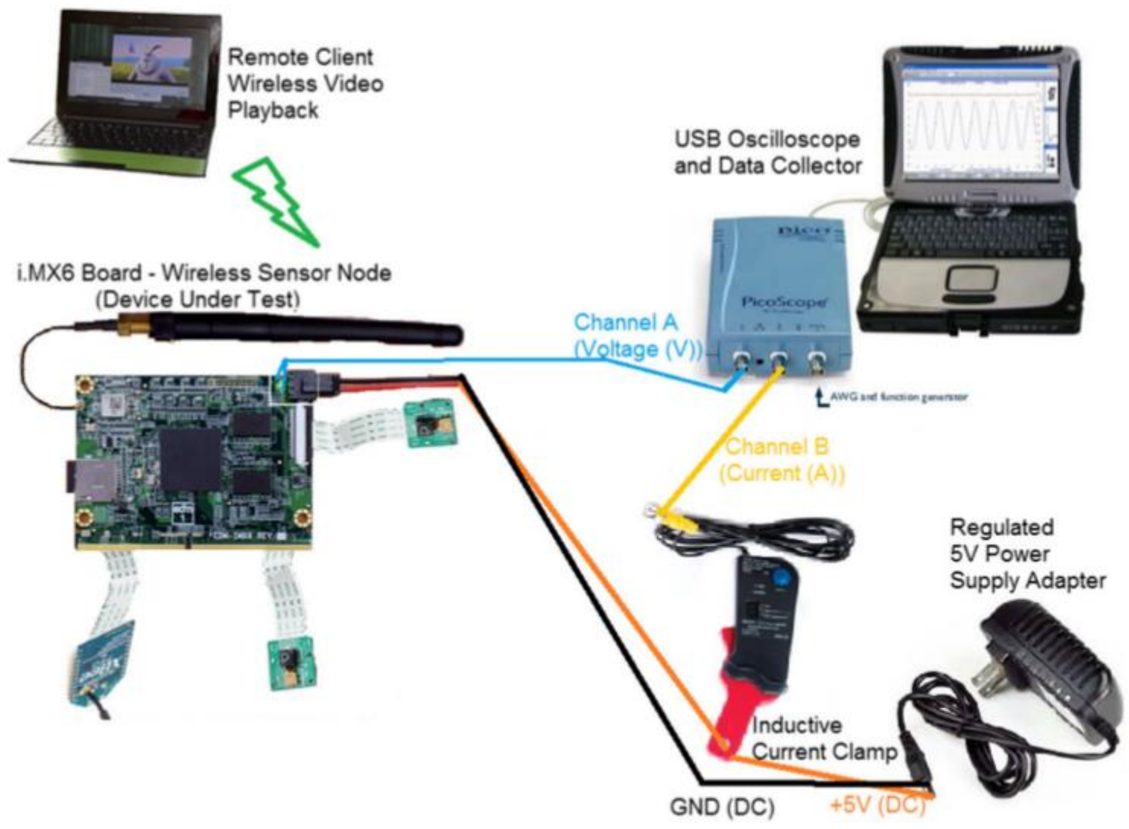


Figure 2.1: Hardware-based Power Measurement Setup [1]

using a plug-in power meter, we will need to recreate the experiment as closely as possible, with the goal of recreating similar current values. Doing so means we can safely test our native capture program with WVSNP-DASH playback to see if we

have successfully saved power. Further detail of the experimental setup that we will conduct as future work is included in Chapter 5.

CHAPTER 3

IMPROVEMENT ON POWER

3.1 Method

We addressed the concerns raised by Seema et al. [1] about their means of measuring the power profile of WVSNP-DASH for miniature wireless multimedia sensor nodes. Their work draws the conclusion that WVSNP-DASH does reduce power consumption on the sensor node for video capture and LIVE streaming compared to the HLS framework. However, the experimental setup for WVSNP-DASH utilized a power-inefficient script that crudely opened and closed the video capture program, instead of a more optimized method. They suggested a native capture program be installed on the camera node itself to further the potential power savings [1]. The contribution of this paper was to create that native application mentioned [1] and run preliminary testing regarding its power savings. Then, we present our findings, discuss the implications, and draft an outline for future testing.

We began by writing a native video capture program using the C language. This program simply takes in several values: name, video length, number of segments, and whether the video is BIG or SMALL (as defined in previous work [1]). We also ensure that the output files are in the WVSNP-DASH format, [2] to ensure they can be read by the WVSNP-DASH player (WDP). Our main limitation is a set time-constraint, but regardless we have optimized the code to the best of our ability. Chapter 3.1 will discuss the performance of the program. Further details of the program are discussed in Chapter 4. Chapter 5 will discuss the need to optimize the program further.

After the program was written, the next step was to establish what factors to measure to determine the efficiency. We compared the performance of our local application to the performance of the script by looking at three key factors: speed, memory usage, and power consumption. All three tests were conducted with laptop testing using Linux-based software to establish a general trend. By establishing this general trend, we establish a benchmark to see if more accurate means of measurement will follow the trend. Next, we analyze the results, suggest techniques for further optimization, and outline how to utilize hardware for more accurate measurements that not only take into account the running process, but also the startup of the device.

For on-laptop testing, we used the following. The laptop used was a *Sager CLEVO P151EMx*. The 2.40 GHz CPU was an *Intel(R) Core™ i7-3630QM* and has 16 GB of RAM. Video capture was done by the internal webcam of the laptop itself. The laptop utilizes a *NB Pro BisonCam*. As for the operating system, we used *Ubuntu* version *16.04.3 LTS*. For the video library, we used FFmpeg version N-89665-gbddf31b. For our measurements, we want the run the script against the native program for 10 minutes. This ensured stable results free from random power spikes or dips. We also ran measurement software in "idle mode," where only minimal operations were running to establish a baseline. To better understand the change in power, we ran the measurement software one minute prior and one minute after running the program or script to show that the system started and finished in the "idle mode" state. We acknowledge that the measurements will be influenced by other operations on the laptop. To minimize their effects, precautions were taken to minimize unnecessary power consumption. First, all networking such as WiFi and Bluetooth were disabled. Any external devices were removed. The laptop brightness was reduced to its minimum value. *Powerstat* was found to be sensitive enough to show mouse and cursor movement, so the laptop was left alone for the duration of testing. And all measurements were done without the laptop being attached to its charger to ensure that the power draw measurements were more accurate. Charging the laptop while measuring power draw can affect the values. Finally, no other programs were running in the background. The only two programs running were the measurement software and either the native program or the script.

For power consumption, *powerstat* [24] was used to create a general trend of power drawn in Watts/s. This Linux tool has been used before [62], [63], [64], [65] to measure the efficiency of wireless sensor nodes, so we felt that the tool would be useful for establishing a general trend. This software tool measures the status of the computer based on a set time interval and the number of samples. The software is even sensitive enough to detect even small movements of the mouse cursor or clicking the left mouse button. The program has additional features beyond power measurements such as time, CPU usage, number or currently running processes, temperature, and more. While previous work [1] measured the results in terms of current in milliamperes, we believe that measuring in watts should be sufficient as both are directly correlated to power consumption. Our results cannot be directly applied to the measurements from the previous work, but if we are able to show a significant reduction in power between the native capture program and the script, then this should be sufficient evidence that the native capture program is appropriate for testing using the hardware test-bed setup from before [1].

For our testing, we set the time interval to 0.5 seconds and the number of samples to 1440, which amounts to 12 minutes of sampling. As mentioned above, *powerstat* was run for one minute to establish a baseline, then the software was run for ten minutes, then the measurement continued for one minute after to ensure that the system returned to the "idle mode." For data collection, we ran 3 runs for each video size and bitrate (BIG and SMALL) and for 3 different time intervals (2 seconds, 5 seconds, and 10 seconds). BIG and SMALL were defined on previous work [1]. BIG video was defined as 500 kilobytes/second with a 640 x 360 pixel resolution and 25 frames/second. SMALL video was defined as 150 kilobytes/second with a 320 x 180 pixel resolution and 15 frames/second. These 3 runs per video size per time length, were conducted twice to ensure consistent values. This testing was done for both the native capture program and the script. To establish a baseline, 3 runs of "idle mode" were run to establish a baseline for each run, making 6 total measurements, or the equivalent of running each case for an hour. To determine the statistical significance of our data and to reject the null hypothesis, a t-test was also run for each case to compare the values of the script and the native capture program. This was done to reinforce the consistency of the *powerstat* measurements. Chapter 3.2 shows the measurements from our runs. The results of the power measurements are in Figure 3.1 and Table 3.1. Chapter 3.3 discusses the results.

Powerstat also showed the CPU usage in percentage, which we used to determine general CPU usage. The tool has a variable called *User* that shows the power consumption of programs initiated by the user [24]. The measurement was done simultaneously to the power draw measurement, so in a similar fashion, the program is idle for one minute before and after its ten minute run, and the results are posted in Figure 3.2 and Table 3.2. Finally, we used the simple Linux tool *time* to show the time lapse of running the program. It is low power and could be run simultaneously with the program itself to ensure accuracy over a more crude method such as a stopwatch. We ran 6 captures of a single segment to obtain the average time to capture a single segment in seconds. We found that the standard deviation of time lapse was very low, so we felt that 6 runs for each condition was significant. The results are displayed in Table 3.3.

3.2 Measurement

Figure 3.2 depicts the resulting power consumption over time of a typical run. Initially, there is a one minute idle period and after ten minutes, we allowed the program to return to the idle state. The results is a square-wave-like shape to show the power consumption. The idle state graph below shows the minimal power consumption when all unnecessary programs are disabled. The program and script are shown as two lines. They account for both sizes and all three video lengths. For all tables, we only took the average and standard deviation of the system when the program was running. We did not include the idle states as they would have falsely reduced the averages and increased the standard deviation. We also conducted a t-test between the power values of the script and the program to determine the statistical significance of our findings. The p values of each run are shown below in each Table. Averages, standard deviation (SD), and p values of each tested case are displayed in Table 3.1. Figure 3.2 shows the plotted power draw in Watts over time.

Table 3.1: Average Power Draw

Size	Time (s)	Program (Watts)	Program SD	Script (Watts)	Script SD	P Value	Idle (Watts)	Idle SD
BIG	2	6.77	1.72	8.13	1.11	$p < 0.0001$	3.93	0.29
	5	7.19	1.80	6.94	1.12	$p < 0.0001$	3.98	0.33
	10	7.68	1.06	6.85	1.14	$p < 0.0001$	3.98	0.33
SMALL	2	5.98	0.57	7.11	0.83	$p < 0.0001$	3.96	0.32
	5	6.03	0.69	6.53	0.87	$p < 0.0001$	3.95	0.34
	10	6.12	0.47	6.86	1.12	$p < 0.0001$	3.92	0.29

Figure 3,2 shows the CPU percent usage. The *powerstat* tool variable called *User* shows the power consumption of programs initiated by the user. Reducing the amount of running programs allows us to determine the general CPU percentage of just the running program. When measuring the idle state, this value is zero. Averages and standard deviations are shown in Table 3.2.

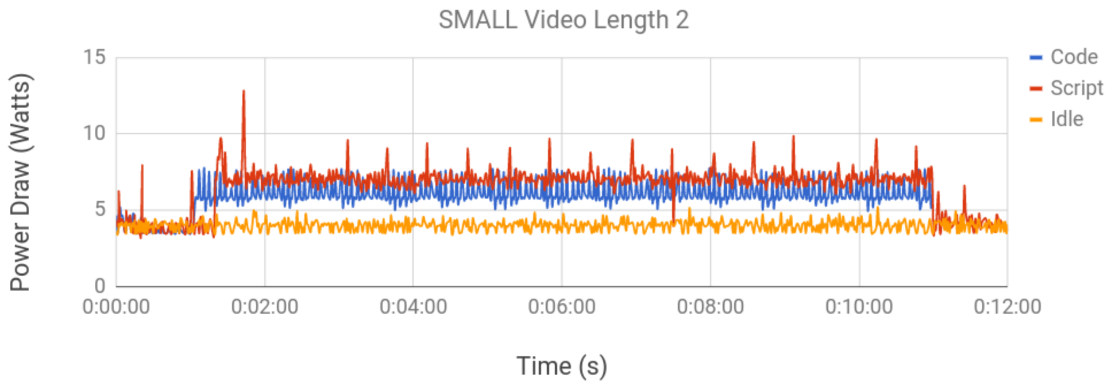
Table 3.2: Average CPU %

Size	Time (s)	Program (%)	Program SD	Script (%)	Script SD	P Value	Idle (%)	Idle SD
BIG	2	2.83	1.34	2.24	0.90	$p < 0.0001$	0.21	0.08
	5	2.67	1.10	2.64	0.82	0.028	0.22	0.11
	10	2.56	0.74	2.52	0.86	0.004	0.22	0.11
SMALL	2	1.97	0.41	1.61	0.62	$p < 0.0001$	0.21	0.14
	5	1.97	0.46	2.46	0.83	$p < 0.0001$	0.25	0.11
	10	1.91	0.47	2.46	0.71	$p < 0.0001$	0.20	0.07

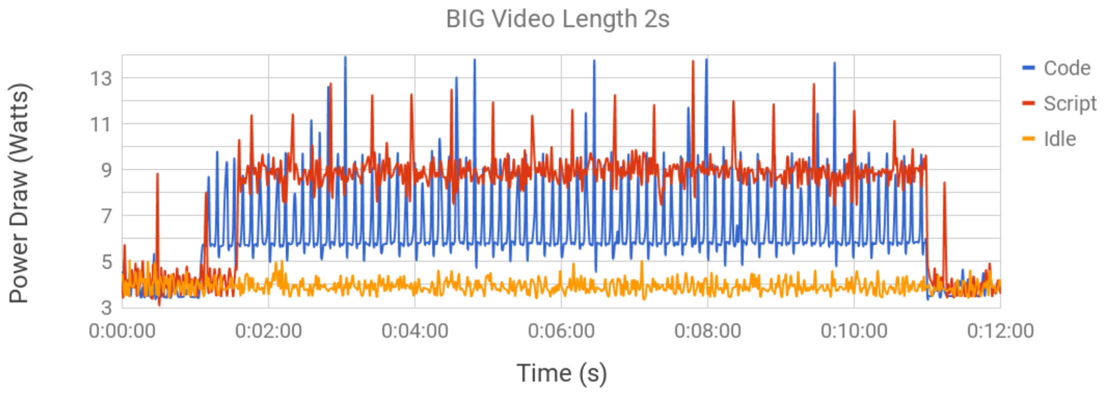
Time lapses of the program were displayed below. The time shown depicts the time taken to record a single time segment of length 2 seconds, 5 seconds, or 10 seconds. Due to the time issue of the native capture program, we reported the average runtimes for different times. Below, Table 3.3 shows the runtimes.

Table 3.3: Average Time Lapse

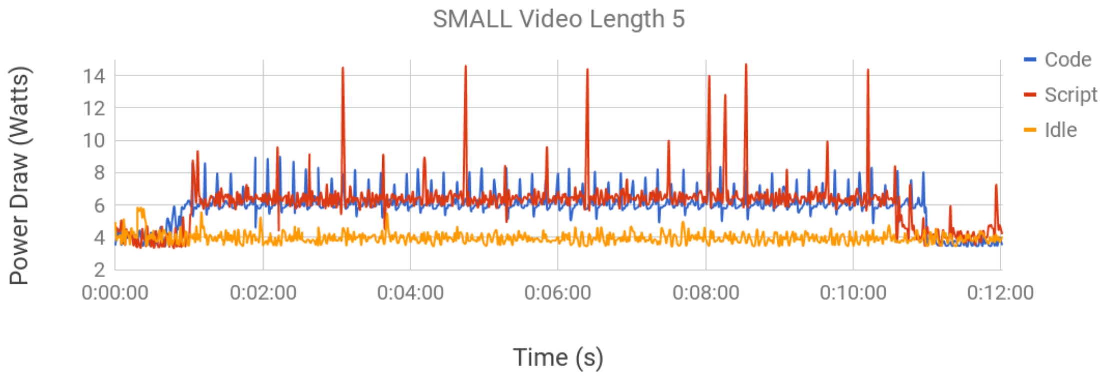
Size	Time	Program (s)	Program SD	Script (s)	Script SD
BIG	2	7.23	0.055	2.22	0.11
	5	17.20	0.015	5.29	0.06
	10	33.90	0.007	10.24	0.18
SMALL	2	4.34	0.003	2.09	0.12
	5	10.36	0.035	5.11	0.09
	10	20.37	0.006	10.24	0.12



(a)

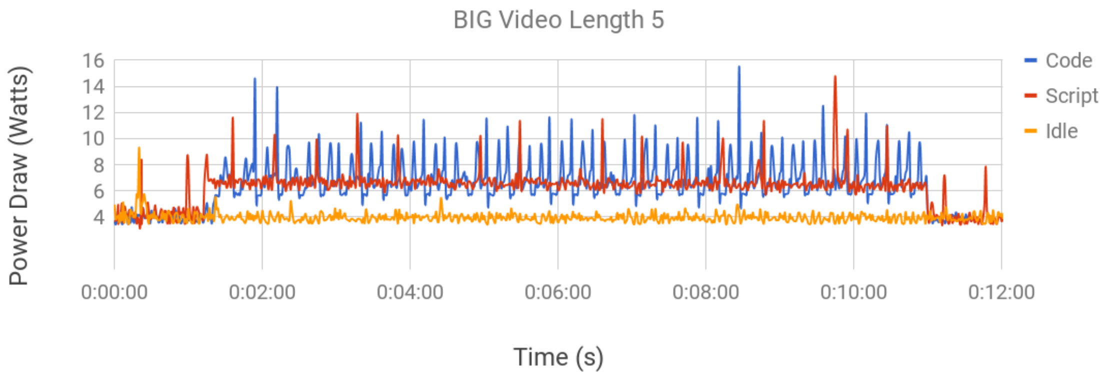


(b)

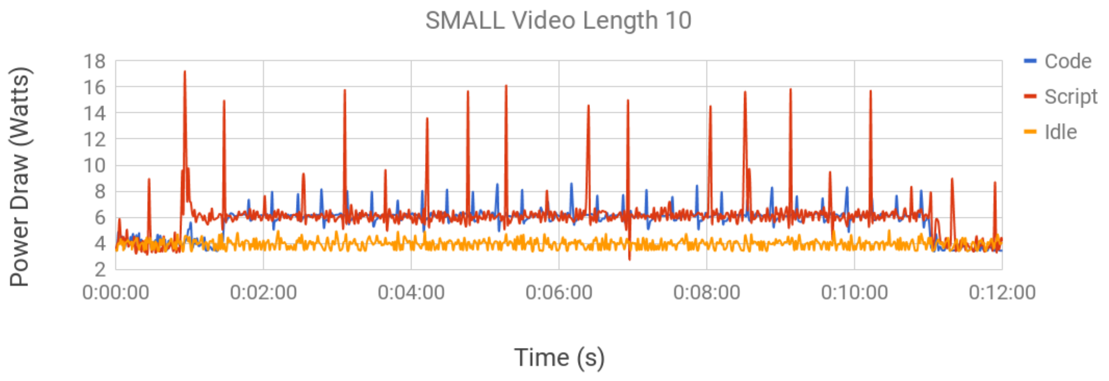


(c)

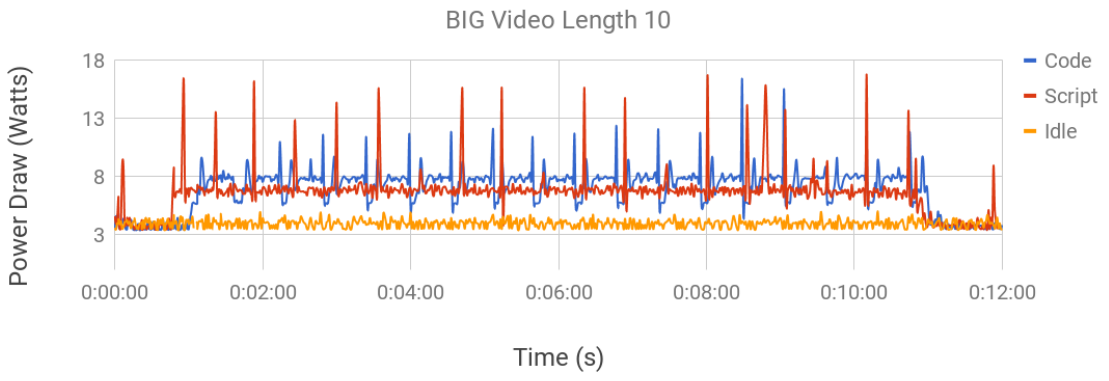
Figure 3.2: Plotted Power Draw Over Time For Each Resolution And Video Segment Length (Continued on Next Page)



(d)

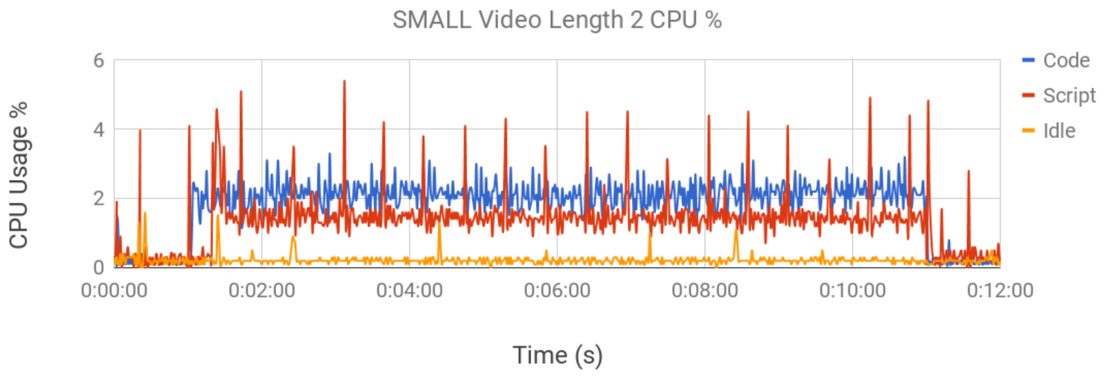


(e)

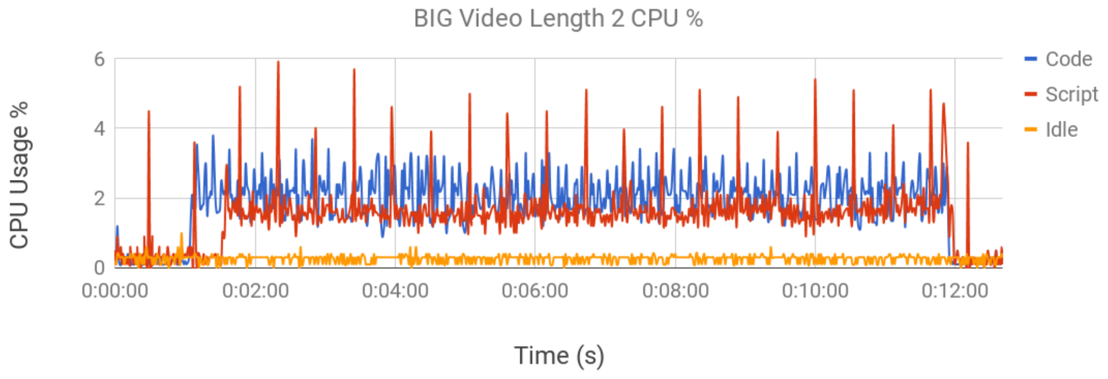


(f)

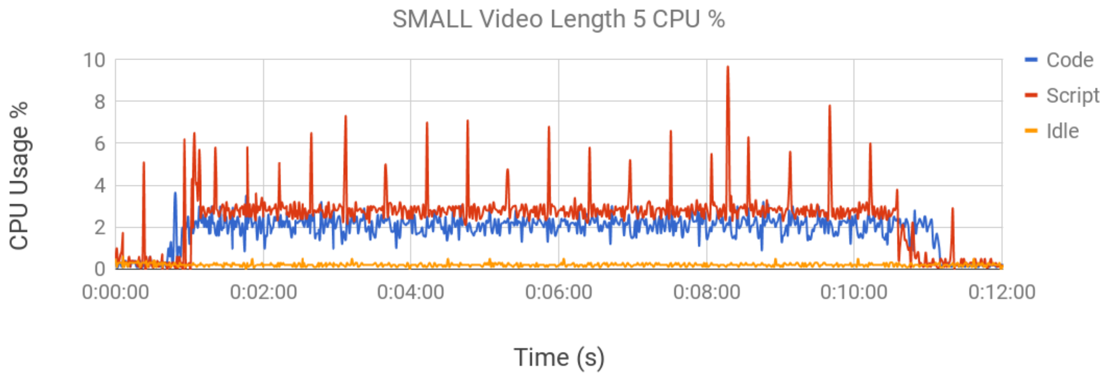
Figure 3.2: Plotted Power Draw Over Time For Each Resolution And Video Segment Length



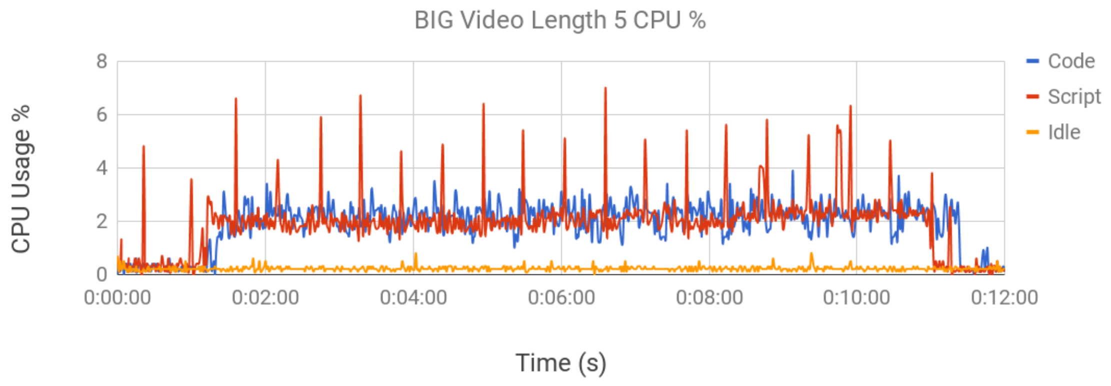
(a)



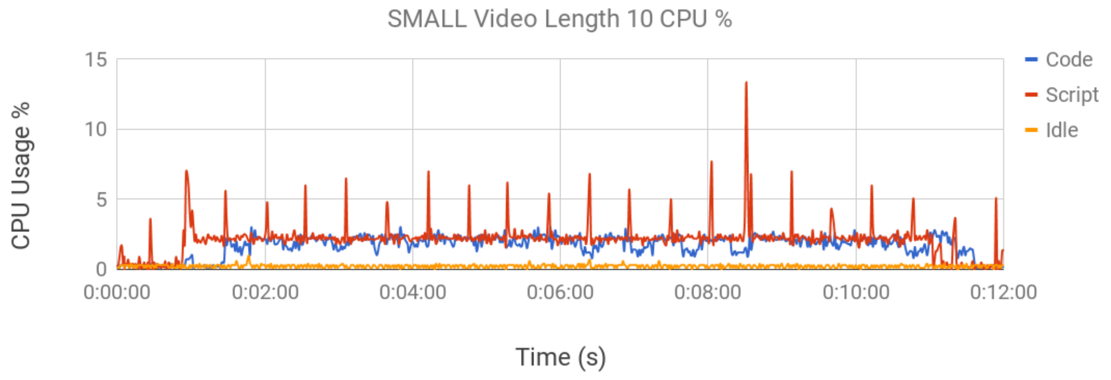
(b)



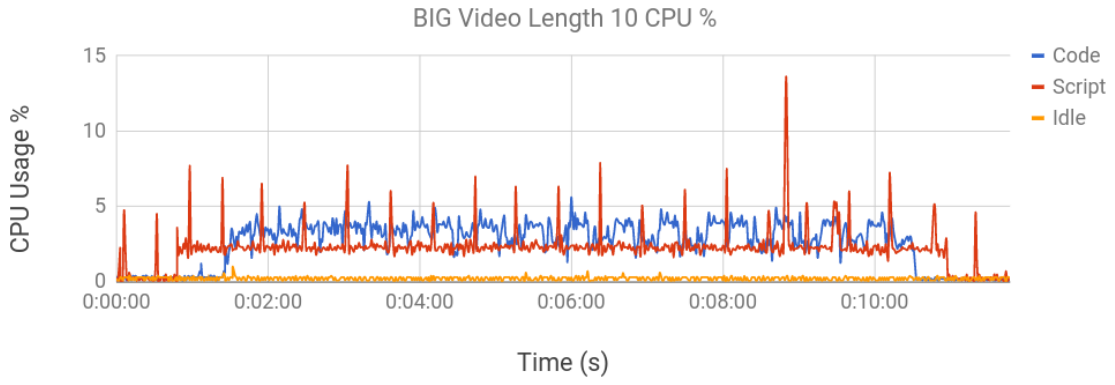
(c)
Figure 3.3: Plotted CPU Usage in Percent Over Time (Continued on Next Page)



(d)



(e)



(f)

Figure 3.3: Plotted CPU Usage in Percent Over Time

3.3 Result

Looking at Table 1, we see that SMALL video saves considerable power over the script on average, especially when recording 2 second segments. For BIG video, there was only savings when recording 2 second video. We double-checked BIG 5 and 10 second segments (Figure 3.1 (d), (f)) with an additional run another time to make sure the result was consistent. Unfortunately, we found that these two conditions still performed the worst. We believe this is due to a power spike at the end of each iteration of the loop. Because our program follows the similar open and close method of the script, unnecessary power usage occurs as FFmpeg is closed and initialized again in the next iteration. It is also likely that BIG video uses more power than SMALL video due to the higher pixel resolution, frame rate, and bit rate. Looking at the graphs, we see that there are times when the native capture program uses less power than the script. The problem comes when opening and closing, where a spike in power increases the overall average power consumption. Future work will need to look at how to reduce power spikes. We also suggest using additional software-based power measurements tools mentioned before [1] to double-check our results.

In regards to CPU percentage, both utilize a similar percentage of the CPU. The native capture program was shown to not cause a spike in CPU usage to the same severity as the script, which is a novel improvement. Once again, SMALL video was shown to be generally more efficient. For future work, we suggest utilizing additional software-based testing means mentioned before [1] to see how much of the CPU processor is used and how much saving is possible there. Both power and CPU of the script vs. the native capture program were found to have $p < 0.05$, in other words, we found the results to be statistically significant.

Time lapse was the greatest issue. Per segment, our native capture program takes twice as long or more as the video length to record. Again, BIG video and longer segments perform worse. There is certainly room for this to be optimized. The program is stable, and consistently performs at the rates in Table 3.3, meaning that the issue is inefficiency in the programming logic. The opening and closing of the software seems to slow down the progress of the video considerably. Yet, playing the segments afterwards shows that they are not affected by the slow recording time. Furthermore, the program shows the elapsed time of recording, which prints at a slower rate than real-time, suggesting the program is rather sluggish. We suggest implementing multithreading to have FFmpeg invoked only once, have the segments generated, and then have FFmpeg closed. In Chapter 4 and 5, we suggest how to further improve the program to get better power consumption, memory usage, and elapsed time per segment.

CHAPTER 4

NATIVE CAPTURE PROGRAM

4.1 How it Works

Before writing the program, several steps were taken to ensure that the program would function properly. First, the latest version of *Ubuntu,* in our case version *16.04.3 LTS*, was installed. Once the operating system was updated, the latest version of FFmpeg was installed. It is important to remember where the various libraries are installed as a path must be configured to this location to ensure that the program functions work properly. As an open-source library, there are multiple helpful tutorials to help understand basic functions of FFmpeg. FFmpeg can be run as a program in the terminal to conduct a variety of video and audio tasks. The script, for example, runs FFmpeg to conduct LIVE video capture. For creating projects in C to call on the FFmpeg libraries, one simply needs to include the libraries at the start of the program as header files. For good examples, locate the “examples” folder inside the FFmpeg folder, as the folder contains multiple example C programs to be used as reference. These range from video to audio encoding and decoding examples and include makefiles and directions.

Next we looked at installing the WVSNP-DASH player and its local server. This folder was provided by the lab of Professor Martin Reisslein at Arizona State University (ASU). Included in the folder are directions to install and operate the WDP, with sample video for playback. Due to time constraints, we were not able to properly conduct LIVE video playback over the WDP using our native capture program, but we intend to do so in the future. To conduct LIVE video playback, we would simply run the local host server included in the WVSNP-DASH folder, open the WDP, and enter the video path to be where the video files are generated. We would run our native capture program. Because our program utilizes name-based segmented video streaming, the player should be able to begin pulling the segments in the correct order based on name.

The native capture program is a C application that utilizes the FFmpeg library for segmented video capture. There are 4 inputs that are required. A sample command-line input would typically look like the following: *./capture --out_fname output.mp4 --vlen 10* -- *seg 60 --size SMALL*. Going through the command-line, *--out_fname* takes in a ".mp4" output name. The file extension must be included. This name will be changed to reflect the WVSNP-DASH format. [2] For example, the example shown above will yield the output: *output-1-0-LIVE-60-1.mp4*, where the filename includes the output name, size, live video, number of segments, and the number of the video file in the sequence (which will gradually increase from 1 to 60 with each subsequent capture).To further accommodate the WVSNP-DASH format, we allowed for two resolutions: BIG and SMALL [1]. These are preset bitrates, pixel resolutions, and framerates defined in Chapter 3, that show in the video name as "1-1" for BIG and "1-0" for SMALL. FFprobe was used to double-check that the resolutions, bitrates, and framerates were correctly set [23].

As stated above, the program utilizes the FFmpeg library for capture and transcoding. This software is highly efficient for encoding and decoding video, and is already used in commercial software like HLS. The previous method for invoking FFmpeg for live capture utilized a script that would open and close ffmpeg. This method is relatively inefficient as additional power was used to interpret the script, launch ffmpeg, conduct the live capture, transcode, and save the files [1]. In contrast, our program aims to invoke ffmpeg once and enters a loop to continually run until the appropriate number of video segments have been created. This is a similar approach to HLS, which uses an optimized method where FFmpeg need only to be invoked once [1].

Our largest contribution was taking a basic open-source video capture program and updating it to work for WVSNP-DASH. The basic video capture program was provided by the lab of Professor Martin Reisslein and Adolph Seema at ASU. We had removed all the deprecated functions, using only the most recent version of FFmpeg and its libraries. This was the largest task we completed, as the program was 3 years out of date in regards to FFmpeg. To remove the deprecated functions, we relied heavily on documentation. While the examples provided by FFmpeg provided an overall picture of what needed to be accomplished, the examples themselves often utilized deprecated functions as well. This meant that we needed to rely more on forum discussions and documentation to determine how to update the program. One good thing about FFmpeg is when compiling the program, the terminal will provide warnings about deprecated functions and often suggest the better implementation technique to use. This made narrowing down the issues much simpler. For future, installing the latest version of FFmpeg and recompiling the native capture program should alert the user if there is a new deprecated function.

Afterward, we created a for loop based on the number of segments desired by the user. The old program was only capable of a single segment capture, so this allowed us to create multiple segments. Finally, we adjusted the file-naming system to be compatible with WVSNP-DASH. While we acknowledge this is not an optimized technique to use, our first goal was to create a WVSNP-DASH compliant program. The result is a program that takes in the above mentioned input, then saves the set number of desired live video segments.

The program works as follows: once the input is taken in, the loop begins. At each iteration, FFmpeg is invoked. First, the audio is read, the appropriate codec is found, and then the audio will be decoded and muxed to the video once the video is read in. For now, all audio input is set to a default synthetic audio. In future, we will consider other audio inputs. Afterwards, we write the filename based on the inputs given. Then, we read in and decode the video file and find the appropriate codec for the video. Here, we set the video input format to live video from the webcam. Once the video stream is read in and the codec is opened, we encode the audio and video, and write to the video frames. At the end of the iteration, we close the operations, free the camera, and reset our values. The process starts again until the desired number of segmented video files have been created.

4.2 Suggestions For Further Optimization

At the moment, the program does not support audio. Rather, it plays a default noise and muxes with the video. Future work can look into audio support, as the code for it has been written, but not fully debugged and implemented. When running the program in the command terminal, the audio source is an optional input, such as the following: *./capture --in_audio_file sound.mp3*. However at the moment, it always returns null, and plays the default sound instead as a temporary measure. Future work should look into fixing this issue. Doing so would allow a user to include either no sound, the default noise, an audio source file, or a microphone recording from the camera node. Considering the IoT application of surveillance, being able to collect audio from the camera node and have it muxed with the live video capture would allow for improved surveillance.

As discussed in Chapter 3, the time lapse has been a major issue. Ideally, a live stream should have a minimal end-to-end delay, but unfortunately, our results in Chapter

3 show that the delay is significant. Utilizing a for loop creates a proportionally longer runtime as the number of segments increases. There is a brief delay at the end of each iteration of the loop that ought to be minimized. In addition, for feedback, the program prints the time in seconds of recording, which to the human observer appears to print significantly slower than real time, meaning the program itself could be optimized further. Another portion of the issue comes from the opening and closing done by the program each run. It was found to be necessary to enable and disable the video capture after each loop to prevent the camera from getting a “in use” error. However, this means that we are fundamentally continuing the same flawed approach done with the script before. While we do save power using the native capture instead, further power can be saved by better understanding what needs to only be invoked once, and keep it outside the for loop, while the for loop creates the sequential segmented live video files. To solve this, we suggest looking at multithreading as the solution. The work ought to be divided so that a portion of the program in one thread generates the video segments and another thread processes new video segment names that are compatible with WVSNP-DASH. Allowing for dynamic naming will allow the user to adjust video quality as the segments are generated. For future work, we intend to invoke FFmpeg only once, then implemented multithreading where the video encoding occurs. This will be further discussed in Chapter 5. We also suggest using FFprobe [23] to double-check that the resolutions, bitrates, and framerates were correctly set after any major change to the program.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Overall, we believe our work was a solid attempt to further optimize native video capture. CPU percent usage was very similar between both pieces of software, but our program did not cause as severe of spikes in CPU usage. This is particularly useful in sensor nodes as they are typically hardware-limited. Power consumption was generally lesser, with the exception of large power spikes that would increase the average power draw in Watts. And finally, time lapse was the largest issue, as it is currently taking much too long to record segments of video versus the current method. For all three observations, BIG video performed significantly worse than SMALL video, most likely due to the higher pixel resolution, bitrate, and framerate. The open and close approach of both our program and the script are wholly inefficient. We know it is possible to improve here as HLS itself utilizes an efficient ffmpeg method where the video library only needs to be invoked once [1].

There were certainly limitations to our work. The largest being time constraints. With the limited time we had to optimize the program, we could only test it to a certain degree, however we feel that the program could be further improved upon. We outline our suggestions in the rest of the paper. The other key limitation is the choice of software-based testing. While we were satisfied with the performance of *powerstat*, we would suggest additional software be used to support the general trends shown in our work. For more rigorous testing, we suggest using alternative power-measurement software as mentioned before, and to utilize the hardware setup shown in Figure 2.1 for a more accurate measurement scheme [1].

We believe the program we have written will be suitable for testing [1] once the following conditions are met. First and foremost, the program itself needs to be further optimized. We discussed in Chapter 4 how to further optimize the program. The program is included in the appendix below as open source software. The main issue is the length of time it takes to capture a segment. We suggest implementing multithreading. We would not have to change the code too drastically. First, we would take in our inputs, then we would invoke FFmpeg only once. The existing for loop would be removed in favor of a smaller one that covers the portion of the program that encodes the video, towards the end of the main function. Here, a second thread would rename the segment once the previous segment is generated. That way, the segments are produced much faster.

For testing, we suggest additional software-based testing ought to be done both with *powerstat* and additional power-measurement libraries to ensure that the optimized native capture program will perform better than the script. For example, we suggest PowerAPI as it had significant literature in support of it [1]. This would be done in isolation to ensure the program itself is properly efficient. Third, we suggest hosting the WVSNP-DASH playback on the testing laptop to measure the power consumption of video playback. This would be done by running the video playback over a local server. This will reaffirm that the native capture program is suitably optimized for WVSNP-DASH, that is has improved performance, and that it is ready for the next phase of testing.

Finally, we outline our intended future actions. We have already set up HLS player to be used for offline purposes, so this phase of testing has some work already done. We hope to retest the work of Seema et. al. [1] using our native capture program for WVSNP-DASH playback against the script and against HLS. This would have us compare WVSNP-

DASH playback using the native capture program and the script, and compare both results to HLS over its own playback service. The same setup as their paper [1] would be assembled, and we would instead measure current draw instead of power draw to properly see the improvement.

Once the software is suitably improved, we would then conduct the hardware measurement. Following the previous work, we will set up the experiment to look very similar to the setup in Figure 2.1 [1]. The main difference is we would use the latest board and camera, though they will generally be the same make. Also, instead of an oscilloscope or current clamp, we would utilize the Mooshimeter to measure our voltage and current [66]. This device is a multimeter that automatically records data over time. Both voltage and current can be recorded simultaneously and the results can be projected on a smartphone. This eliminates the unnecessary waste of having two multimeter attached, and allows us to look at how overall the voltage and current affect the power. Current would be measured by attaching one probe at the power wire. Voltage would be measured at the 5 V jack on the board. Our wireless sensor node would be consist of a *NXP Wandboard QuadPLUS (i.MX6)*, a similar board to the one used in the experiment before [1]. We would attach its matching antennae, and a *5MP Camera Module*. After the board is assembled, we would install the FFmpeg library onto the board and our optimized native capture program. Then, we would setup the WVSNP-DASH playback. We have already setup the WDP and a local server on our testing laptop. The playback can be done over any browser, but for our experiment, we would initially choose Google Chrome. Our test video would be roughly ten minutes of typical surveillance such as a crosswalk on the ASU campus. We would capture 2, 5, 10, and 15 second segment lengths. Our wireless sensor node would host the files to be accessed by the WDP wirelessly over a local server. For measurement, we would begin at a time prior to booting, then we would boot up the node. We would then leave the node active but not run the program to establish a benchmark for the node in an idle state. Then, we would run the program for roughly ten minutes and observe the node returning to the idle state afterwards. We expect each step (except the program running step) to occur for a minute each. This will ensure that our results are easier to compare. Like in the previous experiment, we would focus on milliamperes as our primary independent variable while time would be dependent. Because the board will be maintained at 5 V, we also consider this variable a constant. The primary 3 comparisons would be WVSNP-DASH with the script, WVSNP-DASH with the native capture program, and HLS with its player. We could look at the different video sizes (BIG or SMALL), the different segment lengths (2, 5, 10, and 15s), and additional variables listed in the previous paper [1]. For the different sizes, we would keep it fixed as dynamic streaming would prevent us from seeing the general trend. Each capture would be done with 3 runs at two separate times, like the software testing in this paper, to establish a general trend and minimize variance. What we expect to result from this testing is that WVSNP-DASH with our native capture program will draw the smallest current, followed by WVSNP-DASH with the script, and finally HLS.

Looking at longer-term goals for this project, we suggest further improvements. After testing against HLS, we recommend testing with MPEG-DASH as well, as it is a commonly used streaming framework, and possibly testing against other competition. For the program, we would add audio support as the code has been written but it needs debugging. This would allow for true surveillance. Finally, once the code of the native capture program is optimized, we suggest also looking at the alternative video library GStreamer [55]. This library was also used by Seema et al. [1] and was shown to have even further power savings over FFmpeg. The idea would be to follow the same logic of the FFmpeg-based native capture program, but to incorporate the new libraries instead. Then, we suggest undergoing the same testing done in this paper in Chapter 3 and the future work in Chapter 5 using GStreamer. Due to time-constraints, we were not able to implement a second program, but we believe building it would be worth doing. Furthermore, it has been stated that WVSNP-DASH is not truly dynamic, as video is set as BIG or SMALL [1], [2]. We expect WVSNP-DASH to support more dynamic video streaming in future. When the system is able to support more than the two defined video qualities, the native capture program will need to be adjusted to consider conditions such as limited bandwidth or power saving. Current video frameworks can offer several video qualities or set the value to auto-quality. We hope to see WVSNP-DASH have the capacity to support true dynamic streaming in the future.

Overall, wireless multimedia sensor networks operate in the wider context of access networks, as well as metropolitan area networks that interconnect wireless sensor networks with the Internet at large. An important future work direction is to integrate the WSNP-DASH paradigm with the networking mechanisms into this wider networking context. Generally, there has been recently a trend to control individual network segments as well as their internetworking through Software Defined Networking (SDN) [67], [68], [69], [70]. SDN can enhance the transmissions in wireless networks towards the wired access network segment [71], [72], [73], [74] as well as the management of the wireless networks [75], [76], [77]. Similarly, SDN enhances the wired access network segments

[78], that connect the wireless sensor networks typically through wired, e.g., cable networks [79], [80], [81], [82], [83] or optical fiber networks, e.g., passive optical networks [84], [85], [86], [87], [88], to the corresponding metropolitan area networks. Low-latency DASH service will require short segments. The power saving mechanisms introduced in this thesis should be evaluated for low-latency, short-segment DASH versions in future research. A related research direction is to enhance video surveillance networks to support low-latency, which has become an important requirement in multimedia networking [89], [90], [91], [92], [93], [94].

REFERENCES

[1] A. Seema, T. Shah, L. Schwoebel, Y. Liu, and M. Reisslein, “Power profiling of multimedia sensor node with name-based segment streaming,” Multimedia Tools and Applications, Oct. 2018.

[2] A. Seema, L. Schwoebel, T. Shah, J. Morgan, and M. Reisslein, “WVSNP-DASH: Name-Based Segmented Video Streaming,” IEEE Transactions on Broadcasting, vol. 61, no. 3, pp. 346–355, 2015.

[3] Castellanos WE, Guerri JC, Arce P (2017) “SVCEval-RA: An evaluation framework for adaptive scalable video streaming.” Multimedia Tools Applications. 76(1):437–461.

[4] Chang H-Y (2018) “A connectivity-increasing mechanism of ZigBee-based IoT devices for wireless multimedia sensor networks.” Multimedia Tools Applications. print, pp 1–18.

[5] Hamid Z, Hussain FB, Pyun J-Y (2016) “Delay and link utilization aware routing protocol for wireless multimedia sensor networks.” Multimedia Tools Applications 75(14):8195–8216.

[6] Javaid S, Fahim H, Hamid Z, Hussain FB (2018) “Traffic-aware congestion control (TACC) for wireless multimedia sensor networks Multimedia Tools Applications, in print, pp 1–20.

[7] Kim Y, Bok K, Son I, Park J, Lee B, Yoo J (2017) “Positioning sensor nodes and smart devices for multimedia data transmission in wireless sensor and mobile P2P networks.” Multimed Tools Appl 76(16):17 193–17 211.

[8] Lee J-Y, Jung K-D, Moon S-J, Jeong H-Y (2017) “Improvement on LEACH protocol of a wide-area wireless sensor network.” Multimedia Tools Applications 76(19):19 843–19 860.

[9] Ramakrishna M, Karunakar A (2017) “SIP and SDP based content adaptation during real-time video streaming in future internets.” Multimedia Tools Applications. 76(20):21 171–21 191.

[10] Rashid B, Rehmani MH (2016) “Applications of wireless sensor networks for urban areas: A survey.” J Network Computing Applications 60:192–219.

[11] Shin H, Park J-S (2017) “Optimizing random network coding for multimedia content distribution over smartphones.” Multimedia Tools Applications 76(19):19 379–19 395.

[12] Wunderlich S, Cabrera J, Fitzek F, Reisslein M (2017) “Network coding in heterogeneous multicore IoT nodes with DAG scheduling of parallel matrix block operations.” IEEE Internet Things J 4(4):917–933.

[13] Yap FG, Yen H-H (2014) “A survey on sensor coverage and visual data capturing/processing/ transmission in wireless visual sensor networks.” Sensors 14(2):3506–3527.

[15] Chen S, Yuan Z, Muntean G-M (2016) “An energy-aware routing algorithm for quality-oriented wireless video delivery.” IEEE Trans Broadcast 62(1):55–68.

[16] Kidwai NR, Khan E, Reisslein M (2016) “ZM-SPECK: A fast and memoryless image coder for multimedia sensor networks.” IEEE Sens J 16(8):2575–2587

[17] Pantazis N, Vergados D (2007) “A survey on power control issues in wireless sensor networks.” IEEE Communication Survey Tutorials 9(4):86–107. 4th Quarter.

[18] Popovici E, Magno M, Marinkovic S (2013) “Power management techniques for wireless sensor networks: a review.” In: Proceedings of the IEEE International Workshop on Advance in Sensors and Interface (IWASI), pp 194–198.

[19] Rault T, Bouabdallah A, Challal Y (2014) “Energy efficiency in wireless sensor networks: A top-down survey.” Computing Networks 67:104–122.

[20] Rein S, Reisslein M (2011) “Low-memory wavelet transforms for wireless sensor networks: A tutorial.” IEEE Communication Survey Tutorials 13(2):291–307.

[21] Akyildiz, Ian F., Tommaso Melodia, and Kaushik R. Chowdhury. "Wireless multimedia sensor networks: Applications and testbeds." Proceedings of the IEEE 96, no. 10 (2008): 1588-1605.

[22] Seema, Adolph, and Martin Reisslein. "Towards efficient wireless video sensor networks: A survey of existing node architectures and proposal for a Flexi-WVSNP design." IEEE Communications Surveys & Tutorials 13, no. 3 (2011): 462-486.

[23] Tavli, Bulent, Kemal Bicakci, Ruken Zilan, and Jose M. Barcelo-Ordinas. "A survey of visual sensor network platforms." Multimedia Tools and Applications 60, no. 3 (2012): 689-726.

[24] Chrysafis, Christos, and Antonio Ortega. "Line-based, reduced memory, wavelet image compression." IEEE Transactions on Image processing 9, no. 3 (2000): 378-389.

[25] Oliver, Jose, and Manuel Perez Malumbres. "On the design of fast wavelet transform algorithms with low memory requirements." IEEE Transactions on Circuits and Systems for Video Technology 18, no. 2 (2008): 237-248.

[26] Rein, Stephan, and Martin Reisslein. "Scalable line-based wavelet image coding in wireless sensor networks." Journal of Visual Communication and Image Representation 40 (2016): 418-431.

[27] Rein, Stephan, and Martin Reisslein. "Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks." Ad Hoc Networks 9, no. 4 (2011): 482-496.

[28] Schroeder, Damien, Adithyan Ilangovan, Martin Reisslein, and Eckehard Steinbach. "Efficient multi-rate video encoding for HEVC-based adaptive HTTP streaming." IEEE Transactions on Circuits and Systems for Video Technology (2016). 28(1):143-157, January 2018.

[29] Tausif, Mohd, Naimur Rahman Kidwai, Ekram Khan, and Martin Reisslein. "FrWF-based LMBTC: Memory-efficient image coding for visual sensors." IEEE Sensors Journal 15, no. 11 (2015): 6218-6228.

[30] Ye, Linning, Jiangling Guo, Brian S. Nutter, and Sunanda D. Mitra. "Low-memory-usage image coding with line-based wavelet transform." Optical Engineering 50, no. 2 (2011): 027005.

[31] Sodagar I (2011) "The MPEG-DASH standard for multimedia streaming over the internet." IEEE MultiMed 18(4):62–67.

[32] Thomas E, van Deventer M, Stockhammer T, Begen AC, Champel M-L, Oyman O (2016) "Applications and deployments of server and network assisted DASH (SAND)." In: Proceedings of the IET IBC Conference, pp 1–8.

[33] FFmpeg (2017) [Online]. Available: http://ffmpeg.org

[34] "GStreamer: Open source multimedia framework" (2017) [Online]. Available: http://gstreamer.freedesktop.org/

[35] Engineering Services (2017) Avnet. "WandCam (AES-WCAM-ADPT-G)—getting started guide." [Online]. Available: http://www.em.avnet.com/wandcam

[36] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in Proc. IEEE Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM), Lucca, Italy, 2011, pp. 1–8.

[37] M. Hoernig, A. Bigontina, and B. Radig, "A comparative evaluation of current HTML5 web video implementations," Open J. Web Technol., vol. 1, no. 2, pp. 1–9, 2014.

[38] D. E. Knuth, “Backus normal form vs. Backus Naur form,” Comm. ACM, vol. 7, no. 12, pp. 735–736, Dec. 1964.

[39] A. Molnar and C. H. Muntean, “Cost-oriented adaptive multimedia delivery,” IEEE Trans. Broadcast., vol. 59, no. 3, pp. 484–499, Sep. 2013.

[40] R. Trestian, O. Ormond, and G.-M. Muntean, “Energy-quality-cost trade-off in a multimedia-based heterogeneous wireless network environment,” IEEE Trans. Broadcast., vol. 59, no. 2, pp. 340–357, Jun. 2013.

[41] L. Stevens and R. Owen, “The truth about audio and video in HTML5,” in The Truth About HTML5. New York, NY, USA: Apress, 2014, pp. 117–133.

[42] E. Tzoc and J. Millard, “For video streaming/delivery: Is HTML5 the real fix?” Code4Lib J., vol. 2013, no. 22, Oct. 2013.

[43] S. J. Vaughan-Nichols, “Will HTML5 restandardize the web?” IEEE Computing., vol. 43, no. 4, pp. 13–15, Apr. 2010.

[44] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, “A survey on wireless multimedia sensor networks,” Comput. Netw., vol. 51, no. 4, pp. 921–960, Mar. 2007.

[45] K. Abas, C. Porto, and K. Obraczka, “Wireless smart camera networks for the surveillance of public spaces,” IEEE Comput., vol. 47, no. 5, pp. 37–44, May 2014.354 IEEE TRANSACTIONS ON BROADCASTING, VOL. 61, NO. 3, SEPTEMBER 2015

[46] L. D. P. Mendes, J. J. P. C. Rodrigues, J. Lloret, and S. Sendra, “Cross-layer dynamic admission control for cloud-based multimedia sensor networks,” IEEE Syst. J., vol. 8, no. 1, pp. 235–246, Mar. 2014.

[47] Bicakci K, Gultekin H, Tavli B (2009) “The impact of one-time energy costs on network lifetime in wireless sensor networks.” IEEE Communication Lett 13:12.

[48] Cotuk H, Bicakci K, Tavli B, Uzun E (2014) “The impact of transmission power control strategies on lifetime of wireless sensor networks.” IEEE Trans Computing 63(11):2866–2879.

[49] Cotuk H, Tavli B, Bicakci K, Akgun MB (2014) “The impact of bandwidth constraints on the energy consumption of wireless sensor networks.” In: IEEE Wireless Communications and Networking Conference (WCNC), pp 2787–2792.

[50] Ghasemzadeh H, Panuccio P, Trovato S, Fortino G, Jafari R (2014) “Power-aware activity monitoring using distributed wearable sensors.” IEEE Trans Human-Mach System 44(4):537–544.

[51] Karakus C, Gurbuz AC, Tavli B (2013) “Analysis of energy efficiency of compressive sensing in wireless sensor networks.” IEEE Sens J 13(5):1999–2008.

[52] Li Y, Shen D, Zhou G (2017) “Energy optimization for mobile video streaming via an aggregate model.” Multimed Tools Appl 76(20):20 781–20 797.

[53] Magno M, Boyle D, Brunelli D, Popovici E, Benini L (2014) “Ensuring survivability of resource-intensive sensor networks through ultra-low power overlays.” IEEE Trans Ind Inf 10(2):946–956.

[54] Misra S, Mohanta D (2010) “Adaptive listen for energy-efficient medium access control in wireless sensor networks.” Multimed Tools Appl 47(1):121–145.

[55] Pantazis NA, Nikolidakis SA, Vergados DD (2013) “Energy-efficient routing protocols in wireless sensor networks: A survey.” IEEE Commun Surv Tutorials 15(2):551–591.

[56] Redondi A, Buranapanichkit D, Cesana M, Tagliasacchi M, Andreopoulos Y (2014) “Energy consumption of visual sensor networks: Impact of spatio-temporal coverage.” IEEE Trans Circ Syst Video Technol 24(12):2117–2131

[57] Yan R, Sun H, Qian Y (2013) “Energy-aware sensor node design with its application in wireless sensor networks.” IEEE Trans Instrument Measure 62(5):1183–1191.

[58] Horneber J, Hergenröder A (2014) “A survey on testbeds and experimentation environments for wireless sensor networks.” IEEE Commun Surv Tutorials 16(4):1820–1838.

[59] Saginbekov S, Shakenov C (2016) “Testing wireless sensor networks with hybrid simulators,” arXiv:1602.01567.

[60] Yuan D, Kanhere SS, Hollick M “Instrumenting wireless sensor networks—a survey on the metrics that matter.” Pervasive Mobile Computing 37:45–62. 2017.

[61] Gayan (2012) Powerstat: Power Consumption Calculator for Ubuntu Linux. http://www.hecticgeek.com/ 2012/02/powerstat-power-calculator-ubuntu-linux

[62] Rentala, Sri Harsha, Reisslein, Martin “Analysis of Wireless Video Sensor Network Platforms over AJAX, CGI and WebRTC.” Master’s Thesis. Arizona State University. 2016.

[63] Shah, Tejas, Reisslein (2014) “A Cross-Layer Power Analysis and Profiling of Wireless Video Sensor Node Platform Applications.” Master’s Thesis. Arizona State University.

[64] F.-Q. Sun, G.-H. Yan, X. He, H.-W. Li, and Y.-H. Han, “CPicker: Leveraging Performance-Equivalent Configurations to Improve Data Center Energy Efficiency,” Journal of Computer Science and Technology, vol. 33, no. 1, pp. 131–144, 2018.

[65] Dall, Christopher. (2018) “The Design, Implementation, and Evaluation of Software and Architectural Support for ARM Virtualization.” Columbia University.
[66] Mooshim Engineering. (2018) [Online] Mooshimeter. Available: https://moosh.im/mooshimeter/

[67] Amin, Rashid, and Martin Reisslein, and Nadir Shah. “Hybrid SDN Networks: A Survey of Existing Approaches.” IEEE Communications Surveys & Tutorials, in print, 2018.

[68] Bizanis, Nikos, and Fernando A. Kuipers. "SDN and virtualization solutions for the Internet of Things: A survey." IEEE Access 4 (2016): 5591-5606.

[69] Guck, Jochen W., Amaury Van Bemten, Martin Reisslein, and Wolfgang Kellerer. "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation." IEEE Communications Surveys & Tutorials (2017). 20(1):388-415, First Quarter 2018.

[70] Trois, Celio, Marcos D. Del Fabro, Luis CE de Bona, and Magnos Martinello. "A survey on SDN programming languages: Toward a taxonomy." IEEE Communications Surveys & Tutorials 18, no. 4 (2016): 2687-2712.

[71] Amjad, Muhammad, Fayaz Akhtar, Mubashir Husain Rehmani, Martin Reisslein, and Tariq Umer. "Full-duplex communication in cognitive radio networks: A survey." IEEE Communications Surveys & Tutorials (2017) 19(4):2158-2191, Fourth Quarter 2017.

[72] Kobo, Hlabishi I., Adnan M. Abu-Mahfouz, and Gerhard P. Hancke. "A survey on software-defined wireless sensor networks: Challenges and design requirements." IEEE Access 5 (2017): 1872-1899.

[73] Modieginyane, Kgotlaetsile Mathews, Babedi Betty Letswamotse, Reza Malekian, and Adnan M. Abu-Mahfouz. "Software defined wireless sensor networks application opportunities for efficient network management: A survey." Computers & Electrical Engineering (2017).

[74] Thyagaturu, Akhilesh S., Ziyad Alharbi, and Martin Reisslein. "R-FFT: Function split at IFFT/FFT in unified LTE CRAN and cable access network." IEEE Transactions on Broadcasting (2018).

[75] Ferrari, Lorenzo, Nurullah Karakoc, Anna Scaglione, Martin Reisslein, and Akhilesh Thyagaturu. “Layered Cooperative Resource Sharing at a Wireless SDN Backhaul.”

Proc. IEEE International Conference on Communications Workshops (ICC Workshops), International Workshop on 5G Architecture (5GARCH), pages 1-6, Kansas City, MO, May 2018.

[76] Ndiaye, Musa, Gerhard P. Hancke, and Adnan M. Abu-Mahfouz. "Software defined networking for improved wireless sensor network management: A survey." Sensors 17, no. 5 (2017): 1031.

[77] Thyagaturu, Akhilesh S., Yousef Dashti, and Martin Reisslein. "SDN-based smart gateways (Sm-GWs) for multi-operator small cell network management." IEEE Transactions on Network and Service Management 13, no. 4 (2016): 740-753.

[78] Thyagaturu, Akhilesh S., Anu Mercian, Michael P. McGarry, Martin Reisslein, and Wolfgang Kellerer. "Software defined optical networks (SDONs): A comprehensive survey." IEEE Communications Surveys & Tutorials 18, no. 4 (2016): 2738-2786.

[79] Alharbi, Ziyad, Akhilesh S. Thyagaturu, Martin Reisslein, Hesham ElBakoury, and Ruobin Zheng. "Performance comparison of R-PHY and R-MACPHY modular cable access network architectures." IEEE Transactions on Broadcasting 64, no. 1 (2018): 128-145.

[80] Chen, Po-Yen, and Martin Reisslein. "FiWi network throughput-delay modeling with traffic intensity control and local bandwidth allocation." Optical Switching and Networking 28 (2018): 8-22.

[81] Chen, Po-Yen, and Martin Reisslein. "A simple analytical throughput–delay model for clustered FiWi networks." Photonic Network Communications 29, no. 1 (2015): 78-95.

[82] Hamzeh, Belal, Mehmet Toy, Yunhui Fu, and James Martin. "DOCSIS 3.1: Scaling broadband cable to gigabit speeds." IEEE Communications Magazine 53, no. 3 (2015): 108-113.

[83] Mercian, Anu, Elliott I. Gurrola, Frank Aurzada, Michael P. McGarry, and Martin Reisslein. "Upstream polling protocols for flow control in PON/xDSL hybrid access networks." IEEE Transactions on Communications 64, no. 7 (2016): 2971-2984.

[84] Butt, Rizwan Aslam, Sevia M. Idrus, Nadiatulhuda Zulkifli, and M. Waqar Ashraf. "A Survey of Energy Conservation Schemes for Present and Next Generation Passive Optical Networks." Journal of Communications 13, no. 3 (2018).

[85] McGarry, Michael P., Martin Reisslein, Frank Aurzada, and Michael Scheutzow. "Shortest propagation delay (SPD) first scheduling for EPONs with heterogeneous propagation delays." IEEE Journal on Selected Areas in Communications 28, no. 6 (2010).

[86] Mercian, Anu, Michael P. McGarry, and Martin Reisslein. "Offline and online multi-thread polling in long-reach PONs: A critical evaluation." Journal of Lightwave Technology 31, no. 12 (2013): 2018-2028.

[87] Pradeep, M., B. Pavithra, R. Pooja, S. Parameswari, and M. Pandi. "A Survey of FTTH Elements Based on Broadband Access Network." Asian Journal of Applied Science and Technology (AJAST) 1, no. 7 (2017): 54-59.

[88] Wang, Lin, Xinbo Wang, Massimo Tornatore, Hwan Seok Chung, Han Hyub Lee, Soomyung Park, and Biswanath Mukherjee. "Dynamic bandwidth and wavelength allocation scheme for next-generation wavelength-agile EPON." IEEE/OSA Journal of Optical Communications and Networking 9, no. 3 (2017): B33-B42.

[89] Bachhuber, Christoph, Eckehard Steinbach, Martin Freundl, and Martin Reisslein. "On the minimization of glass-to-glass and glass-to-algorithm delay in video communication." IEEE Transactions on Multimedia 20, no. 1 (2018): 238-252.

[90] Fouladi, Sadjad, Riad S. Wahby, Brennan Shacklett, Karthikeyan Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads." In NSDI, pp. 363-376. 2017.

[91] Nasrallah, Ahmed, Akhilesh Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. "Ultra-Low Latency (ULL) Networks: A Comprehensive Survey Covering the IEEE TSN Standard and Related ULL Research." arXiv preprint arXiv:1803.07673 (2018).

[92] Pandi, Sreekrishna, Frank Gabriel, Juan A. Cabrera, Simon Wunderlich, Martin Reisslein, and Frank HP Fitzek. "PACE: Redundancy engineering in RLNC for low-latency communication." IEEE Access 5 (2017): 20477-20493.

[93] Shih, Yuan-Yao, Wei-Ho Chung, Ai-Chun Pang, Te-Chuan Chiu, and Hung-Yu Wei. "Enabling low-latency applications in fog-radio access networks." IEEE Network 31, no. 1 (2017): 52-58.

[94] Wunderlich, Simon, Frank Gabriel, Sreekrishna Pandi, Frank HP Fitzek, and Martin Reisslein. "Caterpillar RLNC (CRLNC): A Practical Finite Sliding Window RLNC Approach." IEEE Access 5 (2017): 20183-20197.

APPENDIX A

NATIVE CAPTURE PROGRAM SOURCE CODE

```
/*
 * Copyright (c) 2003 Fabrice Bellard
 * Copyright (c) 2016 Adolph Seema
 * Copyright (c) 2018 Zarah Khan
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN
 * THE SOFTWARE.
 */
/**
 * @file
 * Derived from libavformat API example that output a media file in any
 * supported libavformat format. It uses default codecs.
 * Captures live video and saves it to a file.
 *
 */
/*****************************************************************
***************/
/* ZPlayer2 - an ffmpeg-based codec
 * By Zarah Khan
 *
 * Derived from transcoding.c and output.c.
 * To compile:
 * Configure path:  export
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:"$HOME/ffmpeg_build/lib/pkgconfig"
```

```
 * Access the folder with the makefile: type "make"
 * Run the program: ./ZPlayer2 --out_fname tst_out.mp4 --vlen 4
 */
/*********************************************************************
***************/
/*libraries*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <getopt.h>
#include <time.h>
#include <libavutil/avassert.h>
#include <libavutil/channel_layout.h>
#include <libavutil/opt.h>
#include <libavutil/mathematics.h>
#include <libavutil/timestamp.h>

#include <libavformat/avformat.h>
#include <libswscale/swscale.h>
#include <libswresample/swresample.h>
#include <libavdevice/avdevice.h>
/*defintions*/
#define STREAM_FRAME_RATE 25   //25 images per second. perhaps up to 30?
#define STREAM_PIX_FMT    AV_PIX_FMT_YUV420P // default pix_fmt
#define SCALE_FLAGS SWS_BICUBIC // bicubic scaling algorithm -- maybe change
to SWS_BILINEAR? SWS_LANCZOS
//START-------------New Defines-----------------------------------------------
#define LINUX_LIVE_STREAM   // enables linux live stream
#define MAX_NAME_LEN 256    // max length of input?????
typedef struct OutputStream {
  AVFormatContext * video_fmt_ctx;
  AVStream *st;
  AVCodecParameters *str;
  AVCodecContext * video_st;
  AVCodecContext * enc;         // maintains the encoding info
  int64_t next_pts;
  int samples_count;
  AVFrame *frame;
  AVFrame *tmp_frame;
  float t, tincr, tincr2;
  struct SwsContext *sws_ctx;   // swscontext = compile conersion then send to sws_scale
  struct SwrContext *swr_ctx;   // handles audio resampling, sample format conversion,
and mixing
} OutputStream;
```

```
typedef enum {
  AUDIO_SYNTHETIC = 0,
  AUDIO_FILE      = 1,
  AUDIO_MIC       = 2,
  AUDIO_SRC_MAX
} AudioSrcOption;
typedef struct InputStream {
  AVFormatContext * audio_fmt_ctx;    // stores info about the file format
  AVCodecContext * audio_codec_ctx;   // all the codec info from a stream
  AVCodec * audio_codec;              // the audio codec
  AVStream *audio_stream;             // the struct for the stream -- deprecated
  const char *src_filename;           // input
  int audio_stream_idx;               // stores the tyoe of audio 0,1,2
  AVFrame * audio_frame;              //  stores the audio frame
  AVPacket audio_pkt;                 // The struct in which raw packet data is stored.
  int audio_frame_count;              // I din't know
  AudioSrcOption audio_src;           // connects to audio source option
} InputStream;
//END---------------New Defines-----------------------------------------------
//START-------------------------Global variables-------------------------------
char prog_name[MAX_NAME_LEN];         // program name is 256 long
uint32_t video_length = 0; //see command line usage options can be set, I assume
seconds
static int dbg_cnt = 0;               // debugging
uint32_t segment = 0;
char videosize[MAX_NAME_LEN];
// Enable or disable frame reference counting. You are not supposed to support
// both paths in your application but pick the one most appropriate to your
// needs. Look for the use of refcount in this example to see what are the
// differences of API usage between them. */
static int refcount = 0;
//END--------------------------Global variables-------------------------------
// The flush packet is a non-NULL packet with size 0 and data NULL
int decode(AVCodecContext *avctx, AVFrame *frame, int *got_frame, AVPacket *pkt)
{
    int ret;
    *got_frame = 0;
    if (pkt) {
        ret = avcodec_send_packet(avctx, pkt);
        // In particular, we don't expect AVERROR(EAGAIN), because we read all
        // decoded frames with avcodec_receive_frame() until done.
        if (ret < 0)
            return ret == AVERROR_EOF ? 0 : ret;
    }
    ret = avcodec_receive_frame(avctx, frame);
    if (ret < 0 && ret != AVERROR(EAGAIN) && ret != AVERROR_EOF)
        return ret;
    if (ret >= 0)
        *got_frame = 1;
    return 0;
}
int encode(AVCodecContext *avctx, AVPacket *pkt, int *got_packet, AVFrame *frame)
{
  int ret;
  *got_packet = 0;
  ret = avcodec_send_frame(avctx, frame);
  if ((ret < 0) && (ret != AVERROR(EAGAIN)) && (ret != AVERROR_EOF))
    return ret;
  ret = avcodec_receive_packet(avctx, pkt);
  if ((ret >= 0) && (ret != AVERROR(EAGAIN)) && (ret != AVERROR_EOF)){
    //printf("Received packet. ret = %d\n", ret);
    *got_packet = 1;
  }
  if (ret == (AVERROR(EAGAIN)) || AVERROR_EOF){
    return 0;
  }
  return ret;
}
static void log_packet(const AVFormatContext *fmt_ctx, const AVPacket *pkt) {
    AVRational *time_base = &fmt_ctx->streams[pkt->stream_index]->time_base;
    printf("pts:%s pts_time:%s dts:%s dts_time:%s duration:%s duration_time:%s
stream_index:%d\n",av_ts2str(pkt->pts),av_ts2timestr(pkt->pts,time_base),av_ts2str(pkt-
>dts),av_ts2timestr(pkt->dts, time_base),av_ts2str(pkt->duration),av_ts2timestr(pkt-
>duration,time_base),pkt->stream_index);
} //END log_packet(const AVFormatContext *fmt_ctx, const AVPacket *pkt)---------
static int write_frame(AVFormatContext *fmt_ctx,const AVRational
*time_base,AVStream *st,AVPacket *pkt) {
    //printf("In write_frame();\n");
    av_packet_rescale_ts(pkt, *time_base, st->time_base);
    pkt->stream_index = st->index;
    log_packet(fmt_ctx, pkt);
    return av_interleaved_write_frame(fmt_ctx, pkt);  // Write a packet to an output media
file ensuring correct interleaving.
} //END write_frame( ... )-----------------------------------------------------
static void add_stream(OutputStream *ost,AVFormatContext *oc,AVCodec
**codec,enum AVCodecID codec_id, char * dimension) {
  AVCodecContext *c;  // add_stream(&out_video_st,oc,&video_codec,fmt-
>video_codec);
  int ret;
  unsigned int i = 0; // idk if this matters
  *codec = avcodec_find_encoder(codec_id);
  if (!(*codec)) {  //
https://libav.org/documentation/doxygen/master/output_8c_source.html#l00417
      fprintf(stderr, "Could not find encoder for '%s'\n",avcodec_get_name(codec_id));
      exit(1);
  }
  ost->st = avformat_new_stream(oc, *codec);  // AVStream * avformat_new_stream
(AVFormatContext *s, const AVCodec *c)
  if (!ost->st) {
      fprintf(stderr, "Could not allocate stream\n");
      exit(1);
  }
  c = avcodec_alloc_context3(*codec); // old way: c = ost->st->codec;
  if (!c) {
            printf("Could not alloc an encoding context\n");
            av_log(NULL, AV_LOG_ERROR, "Failed to allocate the encoder context for
stream in add_stream: #%u\n", i);
            exit(1);
  }
  ost->enc = c;
  ost->st->id = oc->nb_streams-1;
  ret = avcodec_parameters_to_context(c, ost->st->codecpar);  // sets codec to par
  if (ret < 0) {
            av_log(NULL, AV_LOG_ERROR, "Failed to copy decoder parameters to input
decoder context for stream #%u\n", i);
            exit(1);
  }
  switch ((*codec)->type) {
    case AVMEDIA_TYPE_AUDIO:
      //printf("AVMEDIA_TYPE_AUDIO\n");
      c->sample_fmt  = (*codec)->sample_fmts ?
                    (*codec)->sample_fmts[0] :
                    AV_SAMPLE_FMT_FLTP;
      c->bit_rate    = 64000;
      c->sample_rate = 44100;
      if ((*codec)->supported_samplerates) {
          c->sample_rate = (*codec)->supported_samplerates[0];
          for (i = 0; (*codec)->supported_samplerates[i]; i++) {
              if ((*codec)->supported_samplerates[i] == 44100)
                  c->sample_rate = 44100;
          }
      }
      c->channels      = av_get_channel_layout_nb_channels(c->channel_layout);
      c->channel_layout = AV_CH_LAYOUT_STEREO;
      if ((*codec)->channel_layouts) {
          c->channel_layout = (*codec)->channel_layouts[0];
          for (i = 0; (*codec)->channel_layouts[i]; i++) {
              if ((*codec)->channel_layouts[i] == AV_CH_LAYOUT_STEREO)
                  c->channel_layout = AV_CH_LAYOUT_STEREO;
          }
      }
      c->channels      = av_get_channel_layout_nb_channels(c->channel_layout);
      ost->st->time_base = (AVRational){ 1, c->sample_rate };
      break;
    case AVMEDIA_TYPE_VIDEO:
      //printf("AVMEDIA_TYPE_VIDEO\n");
      c->codec_id = codec_id;
      c->codec_type = AVMEDIA_TYPE_VIDEO; //fixed the codec mismatch problem
      if (strcmp(dimension,"BIG") == 0) { // 25 frames/s
        printf("\nBIG video\n");
        c->width = 640;
        c->height = 360;
        c->bit_rate = 500000; //bits/s
        c->framerate = (AVRational){25,1};  // from encode_video.c
        ost->st->time_base = (AVRational){1, 25};
        c->gop_size = 50;
      }
      else if (strcmp(dimension,"SMALL") == 0) {  // 15 frames/sec
        printf("\nSMALL video\n");
        c->width = 320;
        c->height = 180;
        c->bit_rate = 150000; //bits/s
        c->framerate = (AVRational){15,1};
        ost->st->time_base = (AVRational){1, 15};
        c->gop_size = 30;
      }
      //c->bit_rate = 400000;
      //c->width    = 352;
      //c->height   = 288;
      //ost->st->time_base = (AVRational){1, STREAM_FRAME_RATE};
      c->time_base       = ost->st->time_base;
      //c->gop_size      = 12;
      c->pix_fmt       = STREAM_PIX_FMT;
      if (c->codec_id == AV_CODEC_ID_MPEG2VIDEO) {
          c->max_b_frames = 2;
      }
      if (c->codec_id == AV_CODEC_ID_MPEG1VIDEO) {
          c->mb_decision = 2;
      }
      break;
    default:
      break;
  }
  if ( oc->oformat->flags & AVFMT_GLOBALHEADER ) {
    c->flags |= AV_CODEC_FLAG_GLOBAL_HEADER;
  }
  ost->enc = c;
  ret = avcodec_parameters_from_context(ost->st->codecpar,c);
  if (ret < 0) {
                av_log(NULL, AV_LOG_ERROR, "Failed to copy encoder parameters to
output stream\n");
                exit(1);
  }
}
static AVFrame *alloc_audio_frame(enum AVSampleFormat sample_fmt,uint64_t
channel_layout,int sample_rate, int nb_samples) {
    AVFrame *frame = av_frame_alloc();  // Allocate an AVFrame and set its fields to
default values.
    int ret;
    if (!frame) {
        fprintf(stderr, "Error allocating an audio frame\n");
        exit(1);
    }
    frame->format = sample_fmt;
    frame->channel_layout = channel_layout;
    frame->sample_rate = sample_rate;
    frame->nb_samples = nb_samples;
    if (nb_samples) {
        if ((ret = av_frame_get_buffer(frame, 0)) < 0) {
            fprintf(stderr, "Error allocating an audio buffer\n");
            exit(1);
        }
    }
    return frame;
} //END static AVFrame *alloc_audio_frame( ... )-------------------------------
static void open_audio(AVFormatContext *oc,AVCodec *codec,OutputStream
*ost,AVDictionary *opt_arg) {
  AVCodecContext *c;
  int nb_samples;
  int ret;
  AVDictionary *opt = NULL;
  c = ost->enc;
  if (!c) {
            av_log(NULL, AV_LOG_ERROR, "Failed to allocate the decoder context for
stream\n");
```

```
          exit(1);
  }
  ret = avcodec_parameters_to_context(c, ost->st->codecpar);
  if (ret < 0) {
          av_log(NULL, AV_LOG_ERROR, "Failed to copy decoder parameters to input
decoder context for stream\n");
          exit(1);
  }
  //c = ost->st->codec;                // deprecated -- c = avcodec_alloc_context3(codec);
  av_dict_copy(&opt, opt_arg, 0);      // probably don't need to mess here
  ret = avcodec_open2(c, codec, &opt);  // open the codec
  av_dict_free(&opt);
  if (ret < 0) {
      fprintf( stderr, "Could not open audio codec: %s\n", av_err2str(ret) );
      exit(1);
  }
  //init signal generator
  ost->t     = 0;
  ost->tincr = 2 * M_PI * 110.0 / c->sample_rate;
  //increment frequency by 110 Hz per second
  ost->tincr2 = 2 * M_PI * 110.0 / c->sample_rate / c->sample_rate;

  if (c->codec->capabilities & AV_CODEC_CAP_VARIABLE_FRAME_SIZE)  // c-
>capabilities
      nb_samples = 10000;
  else
      nb_samples = c->frame_size;
  ost->frame     = alloc_audio_frame(c->sample_fmt,c->channel_layout,c-
>sample_rate,nb_samples);
  ost->tmp_frame = alloc_audio_frame(AV_SAMPLE_FMT_S16,c->channel_layout,c-
>sample_rate,nb_samples);
  /* copy the stream parameters to the muxer */
  ret = avcodec_parameters_from_context(ost->st->codecpar, c);
  if (ret < 0) {
    fprintf(stderr, "Could not copy the stream parameters\n");
    exit(1);
  }
  //create resampler context
  ost->swr_ctx = swr_alloc();
  if (!ost->swr_ctx) {
      fprintf( stderr, "Could not allocate resampler context\n" );
      exit(1);
  }
  //set options
  av_opt_set_int( ost->swr_ctx, "in_channel_count", c->channels, 0);
  av_opt_set_int( ost->swr_ctx, "in_sample_rate", c->sample_rate, 0);
  av_opt_set_sample_fmt(ost->swr_ctx,"in_sample_fmt",AV_SAMPLE_FMT_S16,0);
  av_opt_set_int( ost->swr_ctx, "out_channel_count", c->channels, 0 );
  av_opt_set_int( ost->swr_ctx, "out_sample_rate", c->sample_rate, 0 );
  av_opt_set_sample_fmt(ost->swr_ctx,"out_sample_fmt",c->sample_fmt,0);
  //initialize the resampling context
  if ((ret = swr_init(ost->swr_ctx)) < 0) {
     fprintf(stderr, "Failed to initialize the resampling context\n");
     exit(1);
  }
} //END void open_audio( ... )------------------------------------------------
static int decode_packet(InputStream *in_st,int *got_frame,int cached) {
  int ret = 0;
  int decoded = in_st->audio_pkt.size;
  //AVPacket * pkt = in_st->audio_pkt;
  *got_frame = 0;

  if (in_st->audio_pkt.stream_index == in_st->audio_stream_idx) {
    // pCodecCtx WAS allocated first which is required
       ///* THIS LOOKS OKAY FOR NOW
       ret = decode(in_st->audio_codec_ctx,in_st->audio_frame,got_frame,&in_st-
>audio_pkt);
       //*/
       //printf("Finished decoding audio input!\n");
    /*
    //decode audio frame -- deprecated
    ret = avcodec_decode_audio4(  // Decode the audio frame of size avpkt->size from
avpkt->data into frame.
          in_st->audio_codec_ctx,
          in_st->audio_frame, //decoded frame goes in here
          got_frame, //was it really decoded or not goes here
          &in_st->audio_pkt);
    */
    if (ret < 0) {
      fprintf(stderr,"Line#[%d], Error decoding audio frame.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
      return ret;
    //*/
    }
    // Some audio decoders decode only part of the packet, and have to be
    // called again with the remainder of the packet data.
    // Sample: fate-suite/lossless-audio/luckynight-partial.shn
    // Also, some decoders might over-read the packet.
    decoded = FFMIN( ret, in_st->audio_pkt.size );
    if ( *got_frame ) {
      printf("audio_frame[%s] n:%d nb_samples:%d pts:%s\n",(cached ? "(cached)" :
""),in_st->audio_frame_count++,in_st->audio_frame->nb_samples,
        av_ts2timestr(in_st->audio_frame->pts,&in_st->audio_codec_ctx->time_base));
      // Write the raw audio data samples of the first plane. This works
      // fine for packed formats (e.g. AV_SAMPLE_FMT_S16). However,
      // most audio decoders output planar audio, which uses a separate
      // plane of audio samples for each channel (e.g. AV_SAMPLE_FMT_S16P).
      // In other words, this code will write only the first audio channel
      // in these cases.
      // You should use libswresample or libavfilter to convert the frame
      // to packed data.
    } else {
    //decode video frame
    fprintf(stderr,"Line#[%d], we are not expecting video or non audio streams yet. We got
in_st->audio_pkt.stream_index=[%d].\n",
      __LINE__,in_st->audio_pkt.stream_index);
    }
  // If we use frame reference counting, we own the data and need
  // to de-reference it when we don't use it anymore.
    if (*got_frame && refcount) {
    av_frame_unref(in_st->audio_frame);         // frees any reference held by frame
    }
  }
  return decoded;
} //END-decode_packet( ... )--------------------------------------------------
//----------------------------------------------------------------------------
static AVFrame *get_file_audio_frame(InputStream *ist,OutputStream *ost) {
  int got_frame = 0;
  int ret = 0;
  //check if we want to generate more frames
  if (av_compare_ts(
        ost->next_pts,
        ost->enc->time_base,  // ost->st->time_base -----CHANGED HERE
        video_length,
        (AVRational){ 1, 1 }) >= 0) {
    return NULL;
  }
  if (av_read_frame(ist->audio_fmt_ctx, &ist->audio_pkt) >= 0) {
    AVPacket orig_pkt = ist->audio_pkt;
    do {
      if ((ret = decode_packet(ist, &got_frame, 0)) < 0 ) {
        break;
      }
      ist->audio_pkt.data += ret;
      ist->audio_pkt.size -= ret;
    } while (ist->audio_pkt.size > 0);
    av_packet_unref(&orig_pkt);
  }
  //overwriting time stamps with output futures
  ist->audio_frame->pts = ost->next_pts;
  ost->next_pts  += ist->audio_frame->nb_samples;
  return ist->audio_frame;
} //END-static AVFrame *get_file_audio_frame( ... *ist, ... *ost )--------------
//---------------------------------------------------------------------------
static AVFrame *get_synthetic_audio_frame( OutputStream *ost ) {
  AVFrame *frame = ost->tmp_frame;
  int j, i, v;
  int16_t *q = (int16_t*)frame->data[0];
  //check if we want to generate more frames
  if (av_compare_ts(ost->next_pts,
        ost->enc->time_base,  // ost->st->time_base ------ CHANGED
HERE!!!!!!!!!!!!!!!!!!!!!!!!!!
        video_length,(AVRational){ 1, 1 }) >= 0) {
    return NULL;
  }
  for (j = 0; j <frame->nb_samples; j++) {
    v = (int)(sin(ost->t) * 10000);
    for (i = 0; i < ost->st->codecpar->channels; i++)  // ost->st->channels
        *q++ = v;
    ost->t     += ost->tincr;
    ost->tincr += ost->tincr2;
  }
  frame->pts = ost->next_pts;
  ost->next_pts  += frame->nb_samples;
  return frame;
} //END-static AVFrame *get_synthetic_audio_frame(OutputStream *ost)------------
//---------------------------------------------------------------------------
static int write_audio_frame(InputStream *in_st, AVFormatContext *oc, OutputStream
*ost) {
  AVCodecContext *c;
  AVPacket pkt = { 0 }; // data and size must be 0;
  AVFrame *frame;
  int ret;
  int got_packet;
  int dst_nb_samples;
  av_init_packet(&pkt);
  c = ost->enc; /*My attempt to change*/
  ret = avcodec_parameters_to_context(c, ost->st->codecpar);  // should be okay
  if (ret < 0) {
    printf("Could not set codec to paramters\n");
  }  //c = ost->st->codec; // c = avcodec_alloc_context3(in_st->audio_codec);
  if (in_st->audio_src == AUDIO_SYNTHETIC) {  // see above
    frame = get_synthetic_audio_frame(ost);
  } else {
    frame = get_file_audio_frame(in_st,ost);
  }
  if (frame) {
    // convert samples from native format to destination codec format, using
    //the resampler compute destination number of samples.
    dst_nb_samples = av_rescale_rnd(    // Rescale a 64-bit integer with specified
rounding.
                      (swr_get_delay( /*Gets the delay the next input sample will experience
relative to the next output sample. */
                          ost->swr_ctx,c->sample_rate) + frame->nb_samples),c->sample_rate,c-
>sample_rate,AV_ROUND_UP);
    av_assert0(dst_nb_samples == frame->nb_samples);  // CRASHES HERE
    // when we pass a frame to the encoder, it may keep a reference to it internally make
sure we do not overwrite it here
    //simplify -- Ensure that the frame data is writable, avoiding data copy if possible.
    if ((ret = av_frame_make_writable(ost->frame)) < 0) {
      exit(1);
    }
    //convert to destination format -- internal?????
    ret = swr_convert(ost->swr_ctx,ost->frame->data,dst_nb_samples,(const uint8_t
**)frame->data, frame->nb_samples);
    if( ret < 0 ) {
      fprintf(stderr,"Line#[%d], Error while converting.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
      exit(1);
    }
    frame = ost->frame;
    frame->pts = av_rescale_q(ost->samples_count,(AVRational){1, c->sample_rate},c-
>time_base);
    ost->samples_count += dst_nb_samples;
  } //END if (frame)---------------------------------------------------------
  ///*
  //ret = encode(c,&pkt,&got_packet,frame);
  //*/
  //ret = avcodec_encode_audio2( c, &pkt, frame, &got_packet ); //deprecated
  if ((ret = encode(c,&pkt,&got_packet,frame)) < 0 ) {
    fprintf(stderr,"Line#[%d], Error encoding audio frame.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
    exit(1);
  } //encoding audio frame succeeded
  if (got_packet) { //write audio frome to the muxer
```

```
    if ((ret = write_frame( oc, &c->time_base, ost->st, &pkt)) < 0) {
      fprintf(stderr,"Error while writing audio frame: %s\n",av_err2str(ret));
      exit(1);
    }
  } //END if there is a packet, write it to the muxer--------------------------
  return (frame || got_packet) ? 0 : 1;
} //END write_audio_frame(AVFormatContext *oc, OutputStream *ost)---------------
static AVFrame *alloc_picture(enum AVPixelFormat pix_fmt, int width, int height) {
  AVFrame *picture;
  int ret;
  picture = av_frame_alloc();
  if (!picture) {
    return NULL;
  }
  picture->format = pix_fmt;
  picture->width  = width;
  picture->height = height;
  // allocate the buffers for the frame data
  //ret = av_frame_get_buffer(picture, 32);
  if ((ret = av_frame_get_buffer(picture, 32)) < 0) {
    fprintf( stderr, "Could not allocate frame data.\n" );
    exit(1);
  }
  return picture;
} //fine
static void open_video(AVFormatContext *oc,AVCodec *codec,OutputStream *ost,
AVDictionary *opt_arg) {
  AVCodecContext *c;
  int ret;  // open_video(oc,video_codec,&out_video_st,opt);
  c = avcodec_alloc_context3(codec); // done before avcodec_open2
  c = ost->enc;
  if (!c) {
    av_log(NULL, AV_LOG_ERROR, "Failed to allocate the decoder context for
stream\n");
    exit(1);
  }
  AVDictionary *opt = NULL;
  av_dict_copy(&opt,opt_arg, 0);
  ret = avcodec_open2(c,codec,&opt);  // ret = avcodec_open2(c,dec,&opt);
  av_dict_free(&opt);
  if (ret < 0) {
    //printf("ret = %d\n", ret);
    fprintf(stderr, "Could not open video codec: %s\n", av_err2str(ret));
    exit(1);
  }
  ost->frame = alloc_picture(c->pix_fmt, c->width, c->height);
  if (!ost->frame ) {
    fprintf( stderr, "Could not allocate video frame\n" );
    exit(1);
  }
  ost->tmp_frame = NULL;
  if ( c->pix_fmt != AV_PIX_FMT_YUV420P ) {
    ost->tmp_frame = alloc_picture(AV_PIX_FMT_YUV420P, c->width, c->height);
    if ( !ost->tmp_frame ) {
      fprintf( stderr, "Could not allocate temporary picture\n" );
      exit(1);
    }
  }
  ost->video_st = c;
  ost->enc = c;
  /* copy the stream parameters to the muxer */
  ret = avcodec_parameters_from_context(ost->st->codecpar, c);
  if (ret < 0) {
    fprintf(stderr, "Could not copy the stream parameters\n");
    exit(1);
  }
  //printf("Finished open_video\n");
} //END static void open_video(...)--------------------------------------------
#ifdef LINUX_LIVE_STREAM
  //Now lets grab the frame from the input stream
  static void grab_live_image(
    OutputStream *ost,
    unsigned int width,
    unsigned int height,
    AVFormatContext * pFormatCtx,
    AVCodecContext * pCodecCtx,
    AVFrame * pFrame,
    AVPacket * packet,
    int videoindex) {
    int ret, got_picture; //AS-TODO: useful?
    //char buf[1024];
    //AVCodecParserContext * parser;
    // When we pass a frame to the encoder, it may keep a reference to it internally make
sure we do not overwrite it here
    ret = av_frame_make_writable(ost->frame);
    if ( ret < 0 ) {
      exit(1);
    }
    //we must convert LIVE V4L2 picture to the codec pixel format if needed
    if (!ost->sws_ctx) {
      //printf("Setting sws_ctx\n"); /*sws_getContext returns an SwsContext to be used in
sws_scale*/
      //printf("Width = %u and height = %u\n", pCodecCtx->width,pCodecCtx->height);
      //printf("Width = %u and height = %u\n", ost->st->codecpar->width,ost->st-
>codecpar->height);
      /*changed to codecpar -- pix_fmt is not in codecpar, set it to global default*/
      ost->sws_ctx = sws_getContext(pCodecCtx->width,pCodecCtx->height,pCodecCtx-
>pix_fmt,ost->st->codecpar->width,ost->st->codecpar->height,ost->enc-
>pix_fmt,SCALE_FLAGS,NULL,NULL,NULL);
      if (!ost->sws_ctx) {
        fprintf(stderr,"Could not initialize the conversion context\n");
        exit(1);
      }
      //printf("Have sws_ctx\n");
    } //END if (!ost->sws_ctx )-----------------------------------------------
    if(av_read_frame(pFormatCtx, packet) >= 0) {
      if(packet->stream_index == videoindex) {
        //printf("Decoding video...\n"); // Use avcodec_send_packet() and
avcodec_receive_frame(). FIX HERE
        ret = decode(pCodecCtx,pFrame,&got_picture,packet);
        //ret =
avcodec_decode_video2(pCodecCtx,pFrame,&got_picture,packet);  //deprecated!!!!!!!!!!
!!!!!!!!!
        //printf("Finished decoding video input!\n");
        if(ret < 0) {
          printf("Decode Error.\n");
        }
        if(got_picture) {
          //printf("I got picture!\n");
          sws_scale(ost->sws_ctx,(const uint8_t* const*)pFrame->data,pFrame->linesize,0,
            pCodecCtx->height, //ost->st->codec->height,
            ost->frame->data,
            ost->frame->linesize);
        } //END if( got_picture )-------------------------------
      } //END if( packet->stream_index == videoindex )------------
      av_packet_unref(packet); //av_free_packet(packet);
    } //END if( av_read_frame(pFormatCtx, packet) >= 0 )------------------------
    //printf("Grabbed LIVE image\n");
  } //END static void grab_live_image(...)------------------------------------
#else //NOT LINUX_LIVE_STREAM--------------------------------------------------
  ////
  // Prepare a dummy image.
  // This is used only if we want to create a made up frame in the case where
  // we are no capturing live frames nor reading them from some input file or
  // input stream frames.
  ////////////////////////////////////////////////////////////////////////////
  static void fill_yuv_image(AVFrame *pict,int frame_index,int width,int height) {
    int x, y, i, ret;
    // when we pass a frame to the encoder, it may keep a reference to it
    // internally;
    // make sure we do not overwrite it here
    //
    //ret = av_frame_make_writable(pict);
    if ((ret = av_frame_make_writable(pict)) < 0 ) {
      exit(1);
    }
    i = frame_index;
    //IF SYNTHETIC VIDEO
    for ( y = 0; y < height; y++ ) {
      for ( x = 0; x < width; x++ ) {
        pict->data[0][y * pict->linesize[0] + x] = x + y + i * 3;
      }
    }
    //Cb and Cr
    for( y = 0; y < height / 2; y++ ) {
      for (x = 0; x < width / 2; x++) {
        pict->data[1][y * pict->linesize[1] + x] = 128 + y + i * 2;
        pict->data[2][y * pict->linesize[2] + x] = 64 + x + i * 5;
      }
    }

  } //END static void fill_yuv_image(...)---------------------------------------
#endif //#ifdef LINUX_LIVE_STREAM LINUX_LIVE_STREAM
LINUX_LIVE_STREAM-----------
static AVFrame *get_video_frame(
  OutputStream *ost
  #ifdef LINUX_LIVE_STREAM
    ,
    AVFormatContext * in_fmt_ctx,
    AVCodecContext * in_codec_ctx,
    AVFrame * in_frame,
    AVPacket * in_packet,
    int videoindex
  #endif //#ifdef LINUX_LIVE_STREAM
) { //AVCodec *dec = avcodec_find_decoder(ost->st->codecpar->codec_id);
  int ret;
  AVCodecContext *c;// = ost->st->codec; // AVCodecContext *c =
avcodec_alloc_context3(dec);
  c = ost->enc;
  ret = avcodec_parameters_to_context(c, ost->st->codecpar);
  if (ret < 0) {
    av_log(NULL, AV_LOG_ERROR, "Failed to copy decoder parameters to input
decoder context for stream\n");
    printf("FAILED\n");
    exit(1);
  }
  fprintf(stderr,"Line[%d]...next_pts=[%"PRId64 "].\n", __LINE__,ost->next_pts);
  /* check if we want to generate more frames */
 // printf("ost->st->time_base = %s while video_length = %s\n", ost->st->time_base,
video_length);
  if (
    av_compare_ts(ost->next_pts,
        c->time_base, //  ost->st->time_base,
        video_length,(AVRational){ 1, 1 }) >= 0) {
    //printf("Failed here\n");
    return NULL;
  }
  #ifdef LINUX_LIVE_STREAM /*get the image -- see above*/
    //printf("GRAB_LIVE_IMAGE\n");
    grab_live_image(ost,c->width,c-
>height,in_fmt_ctx,in_codec_ctx,in_frame,in_packet,videoindex);
  #else //#ifdef LINUX_LIVE_STREAM
    if ( c->pix_fmt != AV_PIX_FMT_YUV420P ) {
      fprintf(stderr,"Line[%d]..next_pts=[%" PRId64 "].\n", __LINE__, ost->next_pts);
      if ( !ost->sws_ctx ) {
        ost->sws_ctx = sws_getContext(c->width,c->height,AV_PIX_FMT_YUV420P,c-
>width,c->height,c->pix_fmt,SCALE_FLAGS,NULL,NULL,NULL);
        if ( !ost->sws_ctx ) {
          fprintf(stderr,"Could not initialize the conversion context\n");
          exit(1);
        }
      } //END if ( !ost->sws_ctx )-------------------------------------
      fill_yuv_image(ost->tmp_frame,ost->next_pts,c->width,c->height);
      sws_scale(ost->sws_ctx,(const uint8_t * const *)ost->tmp_frame->data,
        ost->tmp_frame->linesize,0,c->height,ost->frame->data,ost->frame->linesize);
    } else {
      fprintf(stderr,"Line[%d]..next_pts=[%" PRId64 "].\n", __LINE__, ost->next_pts );
      fill_yuv_image(ost->frame, ost->next_pts, c->width, c->height);
    }
  #endif
  ost->frame->pts = ost->next_pts++;
  //printf("Finished get_video_frame();\n");
  return ost->frame;
} //END static AVFrame *get_video_frame(...)-----------------------------------
static int write_video_frame(
    AVFormatContext *oc,
    OutputStream *ost
    #ifdef LINUX_LIVE_STREAM
      ,
      AVFormatContext   * in_fmt_ctx,
      AVCodecContext * in_codec_ctx,
      AVPacket * in_packet,
      AVFrame * in_frame,
      int videoindex
    #endif //#ifdef LINUX_LIVE_STREAM
) {
  int ret;
  AVCodecContext *c;
  AVFrame *frame;
  int got_packet = 0;
  AVPacket pkt = { 0 };
  c = ost->enc; //Based on outout.c
  //printf("In video_write_frame\n");
  ret = avcodec_parameters_to_context(c, ost->st->codecpar);
  if (ret < 0) {
    printf("Could not set codec to parameters\n");
  }
  //printf("get_video_frame\n");
  //c = ost->st->codec; // deprecated
  frame = get_video_frame(
            ost,
            #ifdef LINUX_LIVE_STREAM//,
                in_fmt_ctx,
                in_codec_ctx,
                in_frame,
                in_packet,
                videoindex
            #endif //#ifdef LINUX_LIVE_STREAM
          );
  av_init_packet(&pkt);
  //printf("Got video frame. Encoding frame..!\n");
  //ret = avcodec_encode_video2(c,&pkt,frame,&got_packet);  // deprecated
  if ((ret = encode(c,&pkt,&got_packet,frame)) < 0) {
    fprintf(stderr,"Error encoding video frame. ERROR=[%s].\n",av_err2str(ret));
    exit(1);
  }
  //printf("Encoded\n");
  //printf("got packet  = %d\n", got_packet);
  if (got_packet) {
    //printf("got a packet \n"); //%u\n, c->time_base);
    ret = write_frame(oc, &c->time_base, ost->st, &pkt);
  } else {
    ret = 0;
  }
  //printf("ret = %d\n", ret);
  if (ret < 0) {
    fprintf(stderr,"Error while writing video frame. ERROR=[%s].\n",av_err2str(ret));
    exit(1);
  }
  //printf("end write_video_frame\n");
  return (frame || got_packet) ? 0 : 1;
} //END static int write_video_frame(...)--------------------------------------
static void close_stream(AVFormatContext *oc, OutputStream *ost){
  //avcodec_parameters_free(ost->st->codecpar);
  avcodec_close(ost->enc);
  av_frame_free(&ost->frame);          // frees av_frame_alloc()
  av_frame_free(&ost->tmp_frame);
  sws_freeContext(ost->sws_ctx);
  swr_free(&ost->swr_ctx);
} //END close_stream(AVFormatContext *oc, OutputStream *ost)--------------------
// Print information about the input file and the used codec.
static void print_stream_info(InputStream *is) {
  //const char * long_name = NULL;
  //long_name = is->ofmt->long_name;
  fprintf(stderr,"Line#[%d], Codec for input=[%s], is=[%s].\n",__LINE__,
    (is->src_filename? is->src_filename: "NULL"),
    ( // is->audio_stream->codecpar->long_name
      is->audio_codec_ctx->codec->long_name?
      is->audio_codec_ctx->codec->long_name:"NULL"));
  if( is->audio_codec_ctx->codec->sample_fmts != NULL ) {
    fprintf( stderr,"Supported sample formats: ");
    int i = 0;  // is->audio_stream->codecpar->sample_fmts[i]
    for(i = 0; is->audio_codec_ctx->codec->sample_fmts[i] != -1; ++i) {
      fprintf(stderr,"%s",av_get_sample_fmt_name(is->audio_codec_ctx->codec-
>sample_fmts[i]));
      if(is->audio_codec_ctx->codec->sample_fmts[i+1] != -1) {
        fprintf(stderr, ", ");
      }
    }
    fprintf(stderr, "\n");
  }
  fprintf( stderr, "---------\n" );
  fprintf(stderr,"Stream:        %7d\n",is->audio_stream_idx);
  fprintf(stderr,"Sample Format: %7s\n",av_get_sample_fmt_name(is->audio_codec_ctx-
>sample_fmt));
```

```
  fprintf(stderr,"Sample Rate:   %7d\n",is->audio_codec_ctx->sample_rate);
  fprintf(stderr,"Sample Size:   %7d\n",av_get_bytes_per_sample(is->audio_codec_ctx-
>sample_fmt));
  fprintf(stderr,"Channels:      %7d\n",is->audio_codec_ctx->channels);
  fprintf(stderr,"Planar:        %7d\n",av_sample_fmt_is_planar(is->audio_codec_ctx-
>sample_fmt));
  fprintf(stderr,"Float Output:  %7s\n",(av_sample_fmt_is_planar(is->audio_codec_ctx-
>sample_fmt)? "yes" : "no"));
  } //END print_stream_info( ... )---------------------------------------------
//---------------------------------------------------------------------------
// Find the first audio stream and returns its index. If there is no audio
// stream returns -1.%
//---------------------------------------------------------------------------
int find_audio_stream(const AVFormatContext* fmt_ctx) {
  int audio_strm_idx = -1;
  size_t i = 0; // fmt_ctx->streams[i]->codecpar->codec_type
  for( i = 0; i < fmt_ctx->nb_streams; ++i ) {  // Use the first audio stream we can find.
NOTE: There may be more than one, depending on the file.
    if(fmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_AUDIO) {  //
fmt_ctx->codecpar->codec_type
        audio_strm_idx = i;
        break;
    }
  }
  return audio_strm_idx;
} //END find_audio_stream( const AVFormatContext* fmt_ctx )---------------------
//This return "stream_idx" found AND opens the decoder, alloactes it
//and initializes it.
//stream_idx will be "-1" if never found.
static int open_codec_context(InputStream *is, enum AVMediaType type) {
  int ret, stream_index;
  AVStream *st;
  AVCodecContext *dec_ctx = NULL;
  AVCodec *dec = NULL;
  AVDictionary *opts = NULL;
  is->audio_stream_idx = -1; //If AVMediaType is not found this is returned.
  if ((ret = av_find_best_stream( is->audio_fmt_ctx, type, -1, -1, NULL, 0 )) < 0 ) {
    fprintf(stderr,"Could not find [%s] stream in input file [%s].
ERROR=[%s].\n",av_get_media_type_string(type),is->src_filename,av_err2str(ret));
    return ret;
  } else {
    stream_index = ret;
    st = is->audio_fmt_ctx->streams[stream_index];
    //find decoder for the stream
    //dec_ctx = st->codec; // delete this line
    dec = avcodec_find_decoder(st->codecpar->codec_id);  // st->codecpar->codec_id
    dec_ctx = avcodec_alloc_context3(dec);
    //ret = avcodec_parameters_to_context(dec_ctx, st->codecpar);
    if ((ret = avcodec_parameters_to_context(dec_ctx, st->codecpar)) < 0) {
        printf("Failed to load codec parameters to decoder context\n");
        avcodec_free_context(&dec_ctx);
        return ret;
    }
    if (!dec) {
      fprintf(stderr,"Failed to find [%s] codec for stream in input
file=[%s].\n",av_get_media_type_string(type),is->src_filename);
      return AVERROR(EINVAL);
    }
    av_dict_set( &opts, "refcounted_frames", refcount ? "1" : "0", 0 );
    ret = avcodec_open2( dec_ctx, dec, &opts );
    if ( ret < 0 ) {
      fprintf(stderr,"Failed to open [%s] codec.
ERROR=[%s].\n",av_get_media_type_string(type),av_err2str(ret));
      return ret;
    }
    is->audio_stream_idx = stream_index;
  } //found best stream-------------------------------------------------------
  return 0;
} //END-open_codec_context(...)------------------------------------------------
static void print_usage() {
  printf("usage: \n    $ %s --out_fname <string> --vlen <uint32_t>\n"\
    "This program captures LIVE video and outputs it to a media file using
libavformat.\n"\
    "By default, this program generates synthetic audio muxed to a LIVE captured video
stream.\n"\
    "A third option, --audio_source [synth|file|mic], can be used to mux: \n"\
    "  synth => synthetic video + LIVE video (default). \n"\
    "  file  => audio file (e.g. mp3) + LIVE video. \n"\
    "  mic   => default microphone + LIVE video. \n"\
    "\nAgain, the webcam video is encoded and muxed with either\n"\
    "synthetic audio, provided audio file or audio from a microphone.\n"\
    "The video and audio are muxed into a file named in command line.\n"\
    "The output file format is automatically guessed according to the \n"\
    "output file extension.\nRaw images can also be output by using '%%d' in the
filename.\n\n",prog_name);
} //END print_usage()-----------------------------------------------------------
int main( int argc, char **argv ) {
  //float startTime = (float)clock()/CLOCKS_PER_SEC; //
  char *filename = NULL;
  //char ** temp_filename = NULL;
  //char * temp_filename;
  const char *in_audio_file = NULL;
  char * dimension = NULL;
  strncpy( prog_name, argv[0], MAX_NAME_LEN );
  static struct option long_options[] = {
    {"out_fname"   , required_argument, 0, '0' },
    {"vlen"        , required_argument, 0, '1' },
    {"audio_source", optional_argument, 0, '2' },
    {"seg"         , required_argument, 0, '3' },
    {"size"        , required_argument, 0, '4' },
    {0             ,                 0, 0,  0  }
  };
  printf("Line#[%d],dbg_cnt[%d]................................................\n",__LINE__,
dbg_cnt++);
  int capture_options = 0;
  int long_index = 0;
  while ((capture_options = getopt_long(argc,argv,"",long_options,&long_index)) != -1) {
        switch (capture_options) {
                case '0' :
                    filename = optarg;
                    printf("Line#[%d],dbg_cnt[%d]................filename[%s].\n",__LINE__,dbg_c
nt++,(filename == NULL? "NULL": filename));
                    break;
                case '1' :
                    video_length = atoi(optarg);
                    printf("Line#[%d],dbg_cnt[%d]................video_length[%d].\n",__LINE__,d
bg_cnt++,video_length);
                    break;
                case '2' :
                    in_audio_file = optarg;
                    printf("Line#[%d],dbg_cnt[%d]................in_audio_file[%s].\n",__LINE__,d
bg_cnt++,(in_audio_file == NULL? "NULL": in_audio_file));
                    break;
                case '3' :
                    segment = atoi(optarg);
                    printf("Line#[%d],dbg_cnt[%d]................segment[%d].\n",__LINE__,dbg_c
nt++,segment);
                    break;
                case '4' :
                    dimension = optarg;
                    printf("Line#[%d],dbg_cnt[%d]................size[%s].\n",__LINE__,dbg_cnt++
,(dimension == NULL? "NULL": dimension));
                    break;
            default: print_usage();
                exit(EXIT_FAILURE);
      }
  }
  if ( (video_length < (uint32_t)1) || (filename == NULL) ) {
    print_usage();
    exit(EXIT_FAILURE);
  }
  //----------------------ffmpeg related stuff below-------------------------

  //------------------allocate the output media context-----------------------
  for (int k = 1;k<=segment;++k) {
  //temp_filename = filename; // preserve the original name, makes labelling easier
  OutputStream out_video_st = {0}, out_audio_st = { 0 };
  InputStream in_strm = {0};
  AVOutputFormat *fmt;
  AVFormatContext *oc;
  AVCodec *audio_codec, *video_codec;
  int ret;
  int have_video = 0, have_audio = 0;
  int encode_video = 0, encode_audio = 0;
  AVDictionary *opt = NULL;
  av_register_all();
  fprintf(stderr,"Line#[%d],dbg_cnt[%d]......ffmpeg.all.registered.and.ready.to.go!......\n",
__LINE__, dbg_cnt++);
  if ( in_audio_file == NULL ) {
    in_strm.audio_src = AUDIO_SYNTHETIC;
  }
  else {
    in_strm.audio_src = AUDIO_FILE;
    in_strm.src_filename = in_audio_file;
    //open input file, and allocate format context
    if ((ret =
avformat_open_input(&in_strm.audio_fmt_ctx,in_strm.src_filename,NULL,NULL)) < 0)
{
      fprintf(stderr,"Line#[%d], Could not open source file=[%s].
Error=[%s].\n",__LINE__,in_strm.src_filename,av_err2str(ret));
      exit(1);
    } //dump input information to stderr
    av_dump_format( in_strm.audio_fmt_ctx, 0, in_strm.src_filename, 0 );
    //retrieve stream information
    if ((ret = avformat_find_stream_info(in_strm.audio_fmt_ctx, NULL)) < 0) {
      fprintf(stderr,"Line#[%d], Could not find stream information.
Error=[%s].\n",__LINE__,av_err2str(ret));
      exit(1);
    } // Try to find an audio stream.
    in_strm.audio_stream_idx = find_audio_stream(in_strm.audio_fmt_ctx);
```

```
   if( in_strm.audio_stream_idx == -1 ) {  // No audio stream was found.
     fprintf(stderr,"Line#[%d], None of the available [%d streams] are audio
streams.\n",__LINE__,in_strm.audio_fmt_ctx->nb_streams);
     avformat_close_input(&in_strm.audio_fmt_ctx);
     exit(1);
   }
   // open the codec, allocate it, initialize it and return the stream index of the
AVMEDIA_TYPE_AUDIO
   if((ret = open_codec_context(&in_strm, AVMEDIA_TYPE_AUDIO)) != 0) {
     fprintf(stderr,"Line#[%d], Could not open_codec_context.
Error=[%s].\n",__LINE__,av_err2str(ret));
   }
   in_strm.audio_stream =in_strm.audio_fmt_ctx->streams[in_strm.audio_stream_idx];
   if (in_strm.audio_stream == NULL) {
     fprintf(stderr,"Line#[%d], Could not find audio stream in the input=[%s],
aborting!\n"\
     "Use correct audio file source or microphone or default synthetic audio generated by
this program.\n",__LINE__,in_strm.src_filename);
     exit(1);
   }
   //Setup the decoder for the input audio
   ret = avcodec_parameters_to_context(in_strm.audio_codec_ctx, in_strm.audio_stream-
>codecpar);
   if (!audio_codec){
     printf("Failed to copy audio stream parameters to audio codec context\n");
   }
   //in_strm.audio_codec_ctx = in_strm.audio_stream-
>codec;  //in_strm.audio_codec_ctx = in_strm.audio_stream->codecpar;
   in_strm.audio_codec = avcodec_find_decoder(in_strm.audio_codec_ctx-
>codec_id);  // in_strm.audio_stream->codecpar->codec_id
   if( in_strm.audio_codec == NULL ) {
     fprintf(stderr,"Line#[%d], Audio codec not found.\n",__LINE__);
     exit(1);
   }
   print_stream_info(&in_strm);
   //allocate audio frame to be used
   in_strm.audio_frame = av_frame_alloc();
   if (in_strm.audio_frame == NULL) {
     fprintf(stderr,"Line#[%d], Could not allocate frame for input=[%s],
aborting!\n",__LINE__,in_strm.src_filename);
     exit(1);
   }
   //initialize packet, set data to NULL, let the demuxer fill it
   av_init_packet( &(in_strm.audio_pkt) );
   in_strm.audio_pkt.data = NULL;
```

```
    in_strm.audio_pkt.size = 0;
    if (in_strm.audio_stream) {
      fprintf(stderr,"Line#[%d], Demuxing audio from file
input=[%s].\n",__LINE__,in_strm.src_filename);
    }
  printf("Line#[%d],dbg_cnt[%d].REMOVE REMOVE REMOVE! REMOVE
REMOVE! REMOVE REMOVE!\n",__LINE__, dbg_cnt++);
  }
  char c[20];
  char d[20];
  //char e[20];
  size_t len = strlen(filename);
  char * newfilename = malloc (len-3);
  memcpy(newfilename,filename,len-4);
  newfilename[len - 4] = 0;
  printf("newfilename = %s\n", newfilename);
  //strncpy(temp_filename,newfilename,32);
  //temp_filename = newfilename;
  //strncpy(temp_filename,newfilename,MAX_NAME_LEN);
  //snprintf(e,10,"-%s",dimension);
  if (strcmp(dimension,"SMALL")== 0){
    snprintf(d,10,"-1-0-LIVE");//, segment);
  }
  else {
    snprintf(d,10,"-1-1-LIVE");//, segment);
  }
  //snprintf(d,10,"-LIVE-%d", segment);
  snprintf(c,24,"-%d-%d.mp4",segment,k);  // c = "-%d.mp4"
 // strcat(temp_filename,e);
  char * temp_filename = malloc(len-3+strlen(d)+strlen(c));
  printf("just made temp: %s\n", temp_filename);
  memset(temp_filename,0,strlen(temp_filename));
  printf("After memset: %s\n", temp_filename);
  strcat(temp_filename,newfilename);
  strcat(temp_filename,d); //concatenate
  strcat(temp_filename,c); //concatenate
  //strncpy(temp_filename,temp_filename,MAX_NAME_LEN);
  printf("temp_filename = %s\n", temp_filename);
  oc = avformat_alloc_context();
  avformat_alloc_output_context2(&oc,NULL,NULL,temp_filename);  //determines the
file format extension
  if (!oc) {
    fprintf(stderr,"Line#[%d],Could not deduce output format from file extension: using
mp4.\n",__LINE__);
    avformat_alloc_output_context2( &oc, NULL, "mp4", temp_filename); //MPEG
    return 1;
  }
  fmt = oc->oformat;
  if(fmt->video_codec != AV_CODEC_ID_NONE) {
    //printf("No video codec found. Adding stream...\n");
    add_stream(&out_video_st,oc,&video_codec,fmt->video_codec,dimension);  //
working on it
    have_video = 1;
    encode_video = 1;
    //printf("--->Video codec has been found\n");
  }
  if(fmt->audio_codec != AV_CODEC_ID_NONE) {
    //printf("No audio codec found. Adding stream...\n");
    add_stream(&out_audio_st, oc, &audio_codec, fmt->audio_codec, dimension);
    have_audio = 1;
    encode_audio = 1;
    //printf("--->Audio codec has been found\n");
  }
  if(have_video) {
    //printf("Have video. Opening...\n");
    open_video(oc,video_codec,&out_video_st,opt);
    //printf("--->Got the video\n");
  }
  if(have_audio) {
    //printf("Have audio. Opening...\n");
    open_audio(oc,audio_codec,&out_audio_st,opt);
    //printf("--->Got the audio\n");
  }
  av_dump_format(oc,0,temp_filename,1);
  if(!(fmt->flags & AVFMT_NOFILE)) {
    //printf("Output file needed\n");
    if((ret = avio_open(&oc->pb, temp_filename, AVIO_FLAG_WRITE)) < 0 ) {
      fprintf(stderr,"Line#[%d],Could not open file=[%s],
ERROR=[%s].\n",__LINE__,temp_filename,av_err2str(ret));
      return 1;
    }
    //printf("Output file opened\n");
  }
  //printf("Write the header\n");
  if ((ret = avformat_write_header(oc, &opt)) < 0 ) {
    fprintf(stderr,"Line#[%d], Could not write stream header to output file,
ERROR=[%s].\n",__LINE__,av_err2str(ret));
    return 1;
  }
  //printf("Header Written\n");
  //Now let's prepare input video
  #ifdef LINUX_LIVE_STREAM //---------------------------------------------------
    AVFormatContext *pFormatCtx;
    pFormatCtx = avformat_alloc_context();  // create AVFormatContext
    // Since we use V4L2 device make sure all devices are registered
    avdevice_register_all();
    //Linux only
    AVInputFormat *ifmt = av_find_input_format("video4linux2");
    if((ret = avformat_open_input(&pFormatCtx, "/dev/video0", ifmt, NULL)) != 0) {
      fprintf(stderr,"Line#[%d], Could not open input stream.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
      return -1;
    } //Does the stream exist?
    if((ret = avformat_find_stream_info(pFormatCtx,NULL)) < 0 ) {
      fprintf(stderr,"Line#[%d], Could not find input stream infomation.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
      return -1;
    }
    int videoindex = -1;
    int i = 0;
    for(i=0; i < pFormatCtx->nb_streams; i++) {
      if(pFormatCtx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
{
        videoindex = i;
        break;
      }
    }
    if(videoindex == -1) {
      fprintf(stderr,"Line#[%d], Could not find a video stream.
ERROR=[%s].\n",__LINE__,av_err2str(ret));
      return -1;
    } //Setup the decoder for the input video
    AVCodecContext * pCodecCtx; // = pFormatCtx->streams[videoindex]->codec;
    AVCodec        * pCodec;  // = avcodec_find_decoder(pCodecCtx->codec_id);
    pCodec = avcodec_find_decoder(pFormatCtx->streams[videoindex]->codecpar-
>codec_id);
    if (!pCodec) {
      fprintf(stderr, "Could not find input codec\n");
      //avformat_close_input(pFormatCtx);
      return -1;
    }
    /*I think I fixed pCodecCtx*/
    pCodecCtx = avcodec_alloc_context3(pCodec);
    if(!pCodecCtx){
      printf("Failed to allocate pCodecCtx\n");
      return -1;
    }
    if ((ret = avcodec_parameters_to_context(pCodecCtx, pFormatCtx-
>streams[videoindex]->codecpar)) < 0) {
      printf("Could not set codec to paramters\n");
    }
    if(avcodec_open2(pCodecCtx, pCodec, NULL) < 0) {
      fprintf(stderr,"Line#[%d], Could not open codec.
ERROR=[%s]\n",__LINE__,av_err2str(ret));
      return -1;
    }
    AVFrame *pFrame = av_frame_alloc(); // Allocate the input frame to use
    //AVFrame *pFrameYUV = av_frame_alloc();
    AVPacket *packet = (AVPacket *)av_malloc(sizeof(AVPacket));
    //AVPacket * packet = av_packet_alloc();
  #endif //#ifdef LINUX_LIVE_STREAM LINUX_LIVE_STREAM
LINUX_LIVE_STREAM---------
    /*
    if (strcmp(dimension,"BIG") == 0) { // 25 frames/s
      printf("\nBIG video\n");
      pCodecCtx->width = 640;
      pCodecCtx->height = 360;
      pCodecCtx->bit_rate = 500000; //bits/s
      pCodecCtx->framerate = (AVRational){25,1};  // from encode_video.c
      //pFormatCtx->width = 640;
      //pFormatCtx->height = 360;
      //pFormatCtx->bit_rate = 500000; //bits/s
    }
    else if (strcmp(dimension,"SMALL") == 0) {  // 15 frames/sec
      printf("\nSMALL video\n");
      pCodecCtx->width = 320;
      pCodecCtx->height = 180;
      pCodecCtx->bit_rate = 150000; //bits/s
      pCodecCtx->framerate = (AVRational){15,1};
      //pFormatCtx->width = 320;
      //pFormatCtx->height = 180;
      //pFormatCtx->bit_rate = 150000; //bits/s
    }// by default, dimensions will be 640x480
    // pCodecCtx->width = 640;
    // pCodecCtx->height = 480;
    */
  while ( encode_video || encode_audio ) {
  //select the stream to encode
    if (encode_video &&(!encode_audio ||av_compare_ts(
        out_video_st.next_pts,
      out_video_st.enc->time_base,  // changed here
      out_audio_st.next_pts,
      out_audio_st.enc->time_base // changed here
      ) <= 0)) {
  //-------------------------------------------
    encode_video = !write_video_frame(
                oc,
                &out_video_st
                #ifdef LINUX_LIVE_STREAM
                  ,
                  pFormatCtx,
                  pCodecCtx,
                  packet,
                  pFrame,
                  videoindex
                #endif //#ifdef LINUX_LIVE_STREAM
              );
      //printf("encode video\n");
  } else {
    encode_audio = !write_audio_frame(&in_strm,oc,&out_audio_st);
    //printf("encode audio\n");
  }
  //printf("stil in while loop\n");
  } //END while ( encode_video || encode_audio )-------------------------------
//Write the trailer, if any. The trailer must be written before you
  //close the CodecContexts you opened when you wrote the header; otherwise
  //av_write_trailer() may try to use memory that was freed by av_codec_close().
  //printf("write trailer\n");
  av_write_trailer(oc);
  //Close each codec.
  if(have_video) {
    close_stream( oc, &out_video_st );
  }
  if(have_audio) {
    close_stream(oc, &out_audio_st);
  }
  if(!(fmt->flags & AVFMT_NOFILE)) {
    //Close the output file.
    avio_closep(&oc->pb);
  }
  //free the stream
  avformat_free_context(oc);
  avformat_close_input(&pFormatCtx); // kill the camera
  temp_filename = filename; // reset it
}
```

```
  return 0;
} //.............................END.OF.PROGRAM.................................
```