Exploration of the TI AM5728 Audio/Video Subsystem

Dennis Joosens
Faculty of Applied Engineering - Electronics-ICT
University of Antwerp
Antwerp, Belgium
dennis.joosens@student.uantwerpen.be

Theo Debrouwere
Televic
Izegem, Belgium
t.debrouwere@televic.com

Walter Daems
University of Antwerp
Antwerp, Belgium
walter.daems@uantwerpen.be

***Abstract*—Video conferencing systems have been around for several decades and a lot of successful implementations have been developed. A new upcoming trend are High Definition and Ultra High Definition conferencing systems. Nonetheless, these new technologies bring new challenges that have to be overcome. An interesting and important research area in these HD implementations is achieving a small end-to-end latency. For this master's thesis we investigate if it is viable to set up a proof of concept HD video conferencing system that will be used in large conferencing rooms. The communication system needs to attain an overall end-to-end latency beneath 25 ms. In this paper we show that we can send audio over a network and play it back on the receiving end with a minor delay. In contrast to the audio and network part, we are skeptical to keep the video latency below this value. However, the feasibility of the video encoding and decoding needs to be further examined.**

I. Introduction

Over the past decades digital video has become mainstream and is being used in a wide range of applications such as videoconferencing. The last decade HD, Full HD and Ultra HD video conferencing systems are emerging. These higher resolutions generate more data, however the infrastructure does not change that hard so sophisticated techniques need to be applied to cope with these problems. If we would want to send raw video and audio data over a network, we would generate an enormous amount of data. This would cause problems with the memory and storage capacity of many devices and will cause issues with the bandwidth of ubiquitous transmission channels [1]. For instance, we have a screen resolution of 1920x1080 pixels with a frame rate of 30 frames per second and a total of 60 minutes full color video. This would give a data quantity of approximately 671.85 GB. This translates to 11.2 GB per minute and equals a data rate of about 1.49 Gbps. It is needed to use compression and other sophisticated techniques to limit this amount of data and eventually minimize the overall latency.

In this project we need to keep the end-to-end latency beneath 25 ms which is quite a challenge. Nonetheless, there are several successful implementations where the end-to-end latency is beneath 10 ms, so called "zero latency". One of them is the Taos H.264 codec from W&W Communications. The Taos architecture can encode and decode a 1080p60 video simultaneously. It starts encoding and decoding when just a line of pixels is received and does not wait as common codecs for a full frame to start the processing. This way of processing decreases the encode-decode latency to sub 2 ms while not compromising the video quality [2]. We will develop a proof of concept videoconferencing system attaining a latency below 25 ms. For the execution we use the Texas Instruments AM 572x EVM board using the TI AM5728 Sitara™ processor which is based on the ARM® Cortex®-A15 processor. The Sitara processor is a dual core processor clocked at 1.5 GHz, includes two DSP processors, two 3D GPU's and one 2D GPU. The evaluation module itself is based on a BeagleBoard-X15 but extended with a 7 inch LCD capacitive touchscreen, a few buttons and a camera module. However, there have not been any publication or research about this evaluation board, due to the fact that the project is very application specific. For the development we will be using the standard Software Development Kit that is available from Texas Instruments. The SDK has a limited support for codecs that can be used. We will start by measuring the audio, network and video latency and try to form a coherent system.

II. Audio latency

When using real time audio/video applications we do not want to wait for several seconds to interact with the person on the other side. When a latency becomes noticeable for an end user, it can make a conversation troublesome and in some cases even impossible. It is crucial to minimize the overall latency to preserve a good user experience and keep an optimal perceived quality.

Audio latency is declared as the amount of time between a captured sample and the time when the sample is played and is usually measured in milliseconds. In video conferencing systems there are three main components that form the end-to-end latency [3], i.e. the encoding delay that compresses the audio and video data, the network delay and the decoding delay that originates from the decompression process. Low latency is a term that is hard to define. Generally, we can define it as a not disturbing and noticeable latency for the end user. The International Telecommunications Union states that the end-to-end latency should not exceed 100 ms. A delay that small is not perceived by humans [4]. On the other hand, a latency between 100 - 150 ms is also perceived as instantaneous for an end user. When the delay gets higher than 150 ms, it becomes noticeable to the end user and we do not define it anymore as low latency. A delay larger than 300ms is unacceptable and considered distrusting. Other sources say that the end-to-end delay should not be larger than 250 ms [5], [6]. In many cases a small noticeable delay can be acceptable and eventually it all depends on the application. In applications where machines or people interact with video e.g. in the automotive industry, medical systems or video conferencing, requires a very low latency. If we are watching a football game on TV and have an end-to-end latency of 500 ms no one will complain about it. On the other hand, when a ophthalmologist operates a patient using a semi-automatic robot having that amount of delay it would become troublesome.

A first attempt to get an indication of the audio latency, was connecting a microphone and a simple pair of earphones to the EVM board and set up a GStreamer pipeline. We noticed that the delay was audible. This means that the delay using this setup will be certainly greater than 100 ms. The same effect was noticeable on another testbed were we used a laptop with Linux installed on it.

To properly test how an audio system reacts, we look at the impulse response of it. The signal used to test the impulse response of a system is generally a Dirac delta function. However, the Dirac impulse is a hypothetical signal due to the fact that it is infinitely high in amplitude and infinitely small. It can be handy to do calculations but is not applicable in practice. We could also use a periodic signal like a sine wave or block wave. However, due to the periodicity of the input signal, the output signal would be shifted and it would be hard to read the correct latency difference. For the test signal we eventually use a unit step function that we can manually trigger.

As a second attempt to measure the audio latency, we use GStreamer again but now with the finite unit step function as input signal. GStreamer is a pipeline based open source multimedia platform that is used for streaming audio and video content and is implemented in a lot of multimedia players and video editors. A problem that occurred during the measurement was the noisiness of the output channel which made it hard to distinguish the output pulse. We finally noticed that a pulse on the output continuously returned at the same place in time. The measured pulse was attenuated which is due to DSP operations. The latency between the output and input signal is the time difference between the two pulses. The latency is about 225 ms. In figure 1 the pulse circled in red is the input signal, the green circled signal is the delayed output pulse. This amount of delay can be correct due to the use of GStreamer. GStreamer runs on a high level, above ALSA. The Advanced Linux Sound Architecture is the default kernel sound system of Linux. It provides audio and MIDI functionality for the operating system and gives you full control over your sound cards. ALSA contains a kernel and library API, but we will only use the latter. GStreamer adds large buffers to decrease resource usage which creates delays. An audio latency of 225 ms is a lot for a real time system implementation so we needed to look at other solutions to decrease this value.

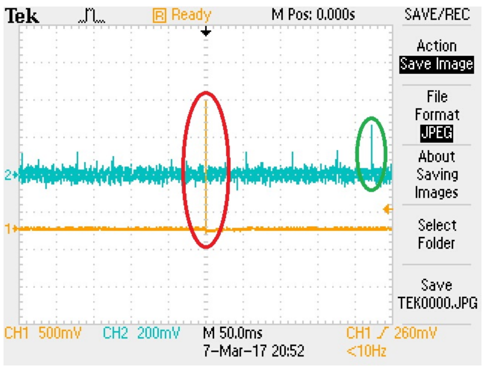

Fig. 1. Latency induced by GStreamer

A third manner to measure the audio latency is to use the latency.c file which is available from the ALSA Library API. The program creates a loopback between the in- and output and calculates the latency difference. Since the EVM board has major differences in processor architecture compared to a desktop or laptop CPU, we needed to cross compile the latency file. The latency.c file is a command line program which gives you the option to apply about a dozen parameters before running the program. These parameters can be the record or playback device, the number of channels to use, the sample frequency, the frame size, the buffer size,. . . . Another advantage is that the ALSA library directly communicates to the ALSA kernel driver which decreases the overall latency. In table I we see a few measurements that have been conducted. We run the program out of the box with no additions or corrections to the code with the exception of using the polling parameter and setting the minimum frame size parameter.

TABLE I
LATENCY WHEN USING THE POLLING PARAMETER AND SETTING A MINIMUM FRAME SIZE

frame size	**latency**
in frames	in milliseconds
256	5.33
512	10.67
1024	21.33
2048	42.67
8192	170.67

Note that the values are calculated and are based on an ideal situation. The latency is dependent on the sample rate and the frame size set. Generally, we can state that the higher the sample rate the lower the latency will be. Also, the smaller the frame size the lower the latency. If we want to know what the latency is generated by the hardware, we have to investigate and compare the input and output signal on an oscilloscope. Since the output default generates a pink noise signal, we need to reconsider our way of measuring.

For the fourth measurement we adapt the latency.c file and insert a signal that we can distinguish easily on the output. The signal can be an impulse or a periodic burst signal with a varying amplitude. The results we will get should be a lot better due to the fact that we are communicating to the hardware on the lowest level possible.

When we change the parameters, and choose a higher sample frequency we get a lower latency. This explains itself, the faster the ADC takes samples of a static dataset, the faster the data will be processed which translates in a smaller latency. The downside is that we need more resources and generate more data. The board itself supports a sample rate up to 96 kHz. However, we choose for 48 kHz and 16 bit samples rather than 96 kHz and 24 bit samples. The reason for this choice is that the last option triples the data rate and needs even more resources. Another reason why we choose 48 kHz is that all hearable frequencies are included and the difference in audio quality will not be noticeable to the end user when choosing an even higher sample rate. We know from the Nyquist theorem that the sampling rate must be at least twice the highest frequency which appears in the audio spectrum. Since the TI board does not support one channel audio we are forced to use two audio channels.

By choosing these values, we can directly calculate the bit rate of this audio stream. The bit rate itself depends on three factors, the amount of audio channels used, the bit depth and the sample frequency. In our case we use two channel audio samples that contain a total of 32 bits per frame. 48000 samples are taken each second. Which results in an uncompressed audio bit rate of 1.536 kbps. Note that we can decrease this rate through the encoding of the raw samples.

Another important parameter that we use is polling. By using polling, ALSA waits for an event, a sample to occur which decreases the load on the Central Processing Unit but slightly increases the latency. If we do not use polling during the running time of the program, one CPU core will run continuously at 100%. When we use the polling parameter, the CPU runs at 1.3-1.4%. There are several other parameters that can have an influence on the latency such as the buffer size and the period size. However, these values all relate to the setting of the frame size and therefore are set automatically based on the minimum latency parameter.

TABLE II
LATENCY MEASURED ON THE HARDWARE

Frame size	**Measured latency**
in frames	in milliseconds
256	6.22
512	11.52
1024	22.20
4096	86.02
8192	171.42

We achieved the lowest latency by using the polling argument and setting the lowest available frame size. The lowest latency we get is 6.22 ms using a frame size of 256. Notice that we use stereo audio and that each sample contains two bytes. This means that each frame exists out of 4 bytes, which means the frame size equals 1024 bytes. The obtained latency is a lot better than the 225 ms we initially had. If we compare the calculated latency of table I to the measured latency on the hardware, we notice that they differ slightly. The software however is based on an ideal situation. It is calculated by dividing the frame size with the sample frequency. The difference can be due to deviations of the sample frequency. But the main reason is that the signal needs to be processed and it takes a certain amount of time to fill and empty the capture device buffer and the playback device buffer.

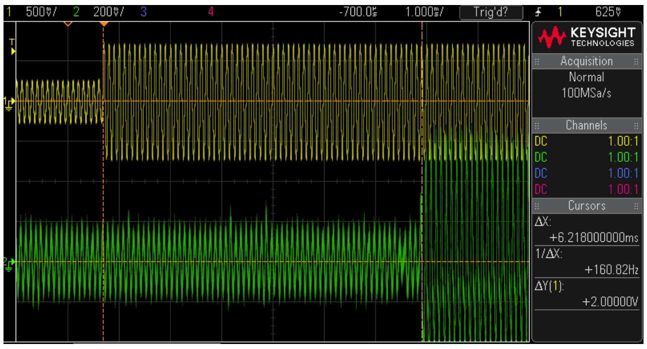

Fig. 2. Smallest obtained delay by using the latency file

If we use polling and increase the minimum buffer size latency parameter, the latency on the output increases immensely. The CPU usage stays between 0.5% and 2% no matter the value of the minimum latency. However when we not use the polling parameter, one CPU core runs at 100% continuously until the program ends. When polling is used, we wait for an event. The events are generated until there is a new period of frames available. However, in the latency file they wait for one second until reading the buffer of the capture device. This simple technique reduces the CPU usage immensely. The signal that is processed is a noisy signal. When we apply the bandpass filter sweep on the signal by adding the effect argument, we still get a noisy output but it contains less spikes. Also changing the minimum latency parameter has no impact under these conditions.

III. Pink Noise

Another measurement we executed was investigating the de facto generated test signal where the latency is calculated from. By default, the latency program generates a signal which can be visualized on an oscilloscope. To know how this signal is generated and how it is used, we need to know what is going on in the code. We found that a pointer to a char type is created which is named buffer. When we dereference this pointer and do a print, we get the actual values. We use a char, which is one byte in length so we will get values in the range of 0 and 255. When we printed these values we indeed get random values within this range. These values give an indication that the output will be a random alternating signal which seems to be noise. In figure 3 we have plotted 100 samples of this generated signal.

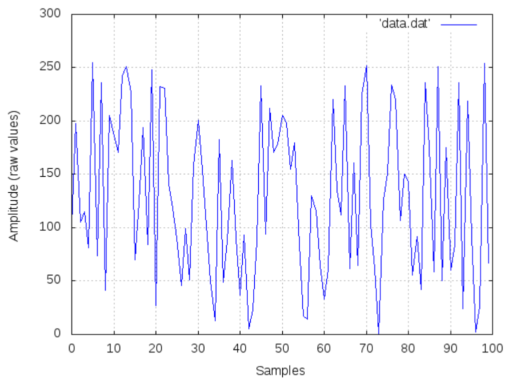

Fig. 3. 100 samples of generated noise

The signal is pink noise which is also known as 1/f noise. It is a random signal that has an equal amount of energy per octave in contrast to white noise which has equal energy per frequency [7]. Pink noise and white noise are closely related. Pink noise is essentially white noise passed through a low pass filter. If we look at the frequency spectrum of white noise we see a uniform distributed flat curve. In comparison with pink noise which has a linearly falling slope that logarithmically decreases with 10dB/decade or 3 dB/octave. This means that every octave contains the same amount of energy and is perceived uniformly by humans. Humans perceive sound in a logarithmic way. This is why pink noise is often used as a reference signal for calibrating and testing audio equipment [7]. The DAC converts these values back into a voltage signal. The end result is a pink noise output signal. It would be impossible to distinguish the latency difference between the output and input if we would use this signal. We would only be able to derive and interpret the calculated latency of the program which acts as a reference point.

IV. Stress Testing

Another important experiment is to examine how the EVM board performs under load. We want to test what effect it has on the audio latency when we stress the available resources. We use the “stress” tool which is available on the board by default. With the stress test we can put load on multiple CPU’s, memory and on I/O. We almost filled up the entire memory and put the two CPU’s on 100% load and examined the effect when different frame sizes where used. In figure 4 we see the output result for a frame size of 256.

We conclude that by executing several stress tests on the system that there is a small impact on the audio latency. The latency increases with a value of about 2 ms no matter the frame size. MacMillan et al. [8] shows that ALSA achieves the lowest latency when the system is under load. Wang et al. [9] confirms that the load on a system has very small effect on the audio processing chain. The experiment proofs that the board is very robust and capable in handling heavy system load.

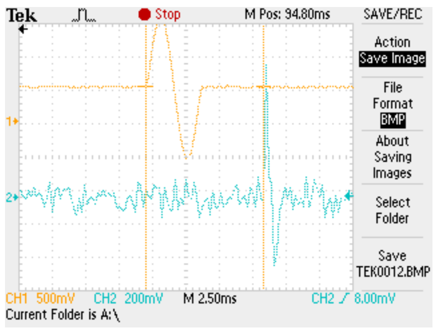

Fig. 4. Smallest latency achieved under stress when using a frame size of 256 and a single sine wave as impulse signal. The orange signal is the input signal, the cyan one is the output.

V. Audio Encapsulation

If we want to send a stream of raw PCM data over a network, we need to slice this data in smaller pieces called segments. The division into segments makes it easier to process and transmit. The OSI layer that is responsible for the segmentation of this data is the transport layer. The transport layer is also responsible for the reassembling of the segments, initiating sessions between hosts and performing flow control and error recovery.

The two most widely used protocols of this layer are the Transmission Control Protocol and the User Datagram Protocol. The two protocols complement each other which means that they both have their pro’s and con’s and their own field of application.

To give a general indication of what the latency of TCP, UDP and on top of that RTP is we have conducted three measurements using the three different encapsulation protocols. We have connected two EVM board to a Gigabit Ethernet switch which was connected to the VM. For the test we calculated the Round-Trip-Time by sending random data and calculating the average RTT value. We have used random data with a length of 5 bytes, 512 bytes and 1024 bytes and conducted 15 measurements per protocol. From table III we notice that the average RTT of TCP is about 64.62% larger than UDP and RTP. This is mainly due to the acknowledgements that the TCP protocol uses to verify that the data has been received successfully and also the retransmission of lost or faulty segments.

A. TCP

TCP is a connection-oriented protocol. It is also a reliable delivery protocol. This means that for every data transfer between hosts a session is created and also terminated properly. The TCP connection establishment happens through the three-way handshake. Note that every segment is

TABLE III
PROTOCOL ROUND-TRIP-TIME OF UDP, TCP AND RTP FOR RANDOM AMOUNTS OF DATA. THE MEASURED VALUES ARE IN MILLISECONDS

	5 bytes			512 bytes			1024 bytes		
Protocol	TCP	UDP	RTP	TCP	UDP	RTP	TCP	UDP	RTP
minimum	0.377	0.141	0.142	0.565	0.159	0.214	0.366	0.187	0.191
maximum	1.201	0.265	0.284	1.150	0.910	0.377	1.483	0.425	0.361
average	0.681	0.203	0.234	0.754	0.276	0.279	0.768	0.285	0.287
stand. dev.	0.189	0.042	0.038	0.120	0.180	0.044	0.272	0.065	0.042

acknowledged by the destination host if received successfully. If this is not the case, the faulty or lost segment will be retransmitted. Another advantage of TCP is that is has built-in functions for flow and congestion control.

B. UDP

In contrast to TCP, UDP is a connection-less protocol. It is also called a best effort protocol. Which means that it does not establish a connection with the destination host first. UDP is a simple protocol that sends out datagrams. These datagrams consist out of an 8-byte header and a payload. Due to its simplicity it does not have built-in functions for flow control or congestion control. Nonetheless, in the header there is an optional field to calculate a UDP checksum. In this way UDP can detect if errors occurred but cannot recover them.

If we look at the headers of the two protocols we see that UDP has a fixed header of only 8 bytes while TCP has a header of 20 bytes which can be extended by 40 optional bytes. Both headers are followed by the payload. These segment headers can be seen as an extra overhead but they are essential for a proper operation of the protocols.

When looking at the overall setup of the two protocols and compare the headers and the extra reliability features of TCP and the lack of them in UDP, we start to notice that UDP has a smaller overall overhead and is better suited for real time applications. In a real time video conferencing system it is unnecessary to retransmit lost or faulty segments. If we would retransmit them it would just give odd effects. Nonetheless, Wheeb shows that UDP has a 22.79% smaller packet delivery ratio than TCP [10]. WHich is due to the lack of acknowledgments in UDP. Aside from that we know that our video conferencing system will be used in a local network so the loss of datagrams and congestion in the network can be marginalized.

C. RTP

On top of TCP and UDP we investigate the Real-time Transport protocol which is another transport layer protocol. However, there is some controversy about it [11]. RTP is essentially an extension to UDP. RTP is included in the payload of an UDP datagram and has a header of 12 bytes which can be extended with extra fields but this is out of scope for our application. RTP is a protocol designed to multiplex several real time data streams into one single stream [11]. Figure 5 displays a complete overview of an RTP packet.

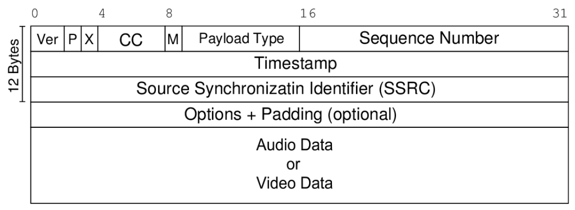

Fig. 5. Overview of the RTP packet.

The most important fields in the RTP header for our application are the following:

- *Payload Type:* This is the type of data that is encapsulated. This can be audio or video data.
- *Sequence number:* The number is incremented for each RTP packet transmitted and can be used to detect packet loss.
- *Timestamp:* The timestamp field shows the sampling instant of the data and needs to be derived from a clock. This field is less important for audio but can become important for video data and synchronization.

Notice that the segments are further encapsulated by the network layer which adds a packet header of 20 bytes and the data link layer adds a frame header of 18 bytes if we exclude the preamble. The size and content of all these different headers can be examined by using the packet analyzing tool Wireshark. We will use RTP for our application because the protocol is optimized for the real time transfer of audio and video which can be derived from the RTP header.

VI. CHANNEL LATENCY

To test the network latency, we adapted the latency file for both the sender and the receiver board. We inserted UDP and additionally RTP functionality which enables us to transfer the data over the network. We will be using RTP since the protocol is designed in such way to efficiently carry real time audio and video data. Another reason is the full support of

RTP to transfer Opus data as payload whcih will be explained in the next section. Figure 6 demonstrates the used lab setup.

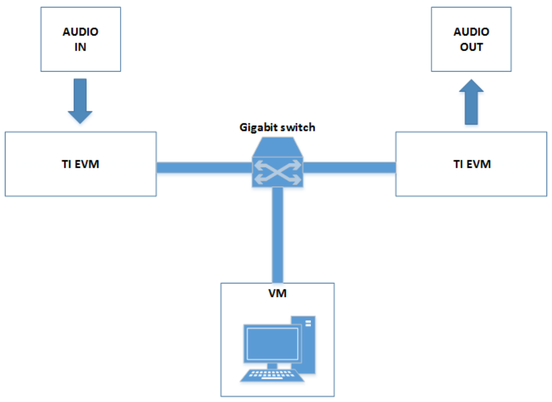

Fig. 6. Lab setup

We are able to send audio in real time over the network by using UDP and RTP. However, the audio that is played back on the receiver EVM board is distorted. Another side effect is the undesirable generation of a periodically high pitched noise that masks the audio. We know that the generated sound is related to the sample frequency. The lower the sample frequency, the lower the frequency of the generated sound.

We were not able to address this issue in time. If we talk into the microphone, we notice that the audio is directly played back on the receiving device. From here on, we can conclude that not a lot of latency will be added. Nonetheless, we did not manage to measure the latency in time.

VII. Audio coding

The EVM board supports AAC and MPEG-2 audio encoding. A major drawback of both codecs is that they generate a latency that is too high for our application. Which means we will not be using these codecs. The Low Delay AAC Profile has a coding delay of 20 ms and the Enhanced Low Delay AAC Profile has a coding delay smaller than 15 ms. They would be ideal for videoconferencing systems but not if you want to achieve a latency below 25 ms.

Another alternative to encode and decode audio is using Opus. Opus is an open source lossy audio codec and is standardized by the Internet Engineering Task Force [12]. It is a hybrid codec and is essentially a combination of Skype's SILK codec and Xiph.Org's CELT codec [13]. The combination of a linear prediction coder with a transform coder has several advantages. The main advantage of Opus is its flexibility. It supports five sampling rates, various bit rates between 6 kbps and 510 kbps, set frame sizes between 2.5 ms and 60 ms, has support for mono and stereo audio and on top of that you can achieve low latencies while maintaining an excellent sound quality [12]. Opus allows the continuous altering of bandwidth, bit rate and even delay when streaming audio over a network [12], [13]. Another advantage is that Opus can be integrated with RTP [13]. The down side of Opus is that the codec is not supported natively by the EVM board. Which means we need to integrate it from scratch.

Opus supports thee operating modes. The SILK-only mode is used for speech, the CELT-only mode is used for music and high-bit rate speech and eventually the Hybrid mode which is a combination of the two codecs. Opus has a 20 ms frame by default that can be changed. When using the SILK-only mode, another 1.5 ms is added for resampling and 5 ms for look-ahead that is required by the SILK layer. All of this gives Opus a default latency of 26.5 ms.

Nonetheless, in CELT-only mode we only need 2.5 ms of look-ahead which gives a total latency of 22.5 ms. The Constrained-Energy Lapped Transform is a speech and music codec that applies a Modified Discrete Cosine Transform and claims to be most efficient at a sampling rate of 48 kHz [14]. We recommend to use the CELT-only mode to meet our latency restriction. We did not manage to implement Opus in time and consider this as future work.

VIII. Future work

The distribution of the audio over the network needs to be optimized and further investigated. We believe we can get a low enough end-to-end latency for the audio and network segment.

The implementation and measuring of the video latency needs to be executed and further investigated. We suggest to start from the H.264 standard, due to the fact that it is fully supported by the EVM board. Another reason to use H.264 as a starting point is the availability of low latency profiles in the standard. We would recommend using the Baseline Profile or the Constrained Baseline Profile which are both targeted for real time communication systems. Both profiles eliminate the B-frames and use only I and P frames. In this way, H.264 claims to support real time and low latency applications. However, low latency is a relative concept in these profiles and will still be higher than the restricted 25 ms.

The design goal of the H.264 video compression standard was to reduce the bandwidth while maintain the same video quality, not to achieve the lowest latency. Nonetheless, there are several successful sub zero video encoding and decoding implementations. Many implementations are based on the H.264 e.g. Taos, while other implementations use completely adapted versions of the H.264 standard and run on top of specifically designed FPGA based "ASIC" chipsets.

We are forced to do several adaptations to the internals of the H.264 standard, since the latency of a encoded frame in the native H.264 standard is high.

In traditional codecs the encoding process starts when a complete frame of video is present. If we have a frame rate of 30 fps, this will result in 33.3 ms of latency from the encoder and another 33.3 ms at the decoder. Which results in a total latency of at least 66.67 ms. We remain skeptical if these modifications to the the encoder and decoder will suffice to stay under 25 ms. we could also increase the frame rate to decrease the latency. This will not solve the problems completely and it will generate even more data and would need more processing power.

IX. Conclusion

In this paper we have tried to develop and setup a proof-of-concept video conferencing system. First we started by investigating and measuring the audio latency by applying several measurement techniques. We have analyzed the by default provided manufacturers testing signal, pink noise and tested the effect on the audio latency when the system was under load.

After that, we have studied the transport layer protocols TCP, UDP and RTP and showed that RTP will be the protocol of choice to encapsulate the audio data.

As a result, we have set up a local audio conferencing system and showed it is viable to send audio data in real time while keeping the latency low. We shortly described and advised to use the Opus codec for audio encoding and decoding.

Further, we recommend to use the H.264 standard as a starting point for the video coding segment. We think that it is not possible to develop this system with an end-to-end latency that low. It should be possible for audio but we will run into problems when encoding and decoding video. Although, the video encoding and decoding implementation and measurement needs to we conducted. We believe that a total real time stream can be achieved having an end-to-end latency with a value between 60 and 100 ms.

References

[1] S. Ponlatha and R. S. Sabeenian, “Comparison of Video Compression Standards,” *International Journal of Computer and Electrical Engineering*, vol. 5, no. 6, pp. 549–566, 2013.

[2] Taos, “Taos - A Revolutionary H.264 Video Codec Architecture For 2-Way Video Communications Applications,” 2008.

[3] R. M. Schreier and A. Rothermel, “A Latency Analysis on H.264 Video Transmission Systems,” *2008 Second International Conference on Electrical Engineering*, pp. 1–2, 2008.

[4] M. M. Al-n, “A Guideline to Video Codecs Delay,” vol. 4, no. 2, pp. 5–8, 2014.

[5] S. Rowshanrad, S. Namvarasl, B. Jamasb, and M. Keshtgary, “Video Codec Standards Comparison for Video Streaming Over SDN,” vol. 5, no. 1, pp. 10–15, 2015.

[6] J. Wu, J. Yang, X. Wu, and J. Chen, “A Low Latency Scheduling Approach for High Definition Video Streaming over Heterogeneous Wireless Networks,” *Globecom2013-Communication Software, Services and Multimedia Symposium*, no. December, pp. 1723–1729, 2013.

[7] G. Ballou, *Handbook for Sound Engineers 4th Edition*, ser. Focal Press. Focal, 2013.

[8] K. MacMillan, M. Droettboom, and I. Fujinaga, “Audio latency measurements of desktop operating systems,” *Proceedings of the International Computer Music Conference*, pp. 259–262, 2001.

[9] Y. Wang, R. Stables, and J. Reiss, “Audio latency measurement for desktop operating systems with onboard soundcards,” *New York*, pp. 1–10, 2010.

[10] A. H. Wheeb, “Performance Comparison of Transport Layer Protocols,” *International Journal of Advanced Research in Computer Science and Software Engineering 5(12),December-2015*, no. December 2015, pp. 121–125, 2015.

[11] A. Tanenbaum and D. Wetherall, *Computer Networks Fifth Edition, International Edition*. Pearson Prentice Hall, 2011.

[12] J. Valin, K. Vos, and T. Terriberry, “RFC 6716 Opus Definition,” *Internet Engineering Task Force (IETF)*, p. 326, 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6716

[13] J.-M. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, “High-Quality, Low-Delay Music Coding in the Opus Codec,” *135th AES Convention*, pp. 73–82, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84896468761{\&}partnerID=tZOtx3y1

[14] T. Ogunfunmi, R. Togneri, and M. Narasimha, *Speech and Audio Processing for Coding, Enhancement and Recognition*, ser. EBL-Schweitzer. Springer New York, 2014.