CCTV Camera Software Customization for Precise Frame Capture Time Retrieval

Serhii Matsyshyn
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
serhii.matsyshyn@ucu.edu.ua

Oleksii Mytnyk
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
oleksii.mytnyk@ucu.edu.ua

Tymur Krasnianskyi
Faculty of Applied Sciences
Ukrainian Catholic University
Lviv, Ukraine
tymur.krasnianskyi@ucu.edu.ua

***Abstract*—We present a series of steps that allow for the injection of high-precision timestamps. Most CCTV cameras do not provide an absolute timestamp of the frame, which is crucial in solving real-time computer vision tasks. A common approach for timestamp estimation is to subtract the average delay from the frame retrieval time on the client's side. However, the delay accumulates from many stages of image processing and varies substantially. Whereas our approach is to inject timestamps into the video stream on the camera's side. Our customized software provides highly accurate results: the standard deviation of the difference between predicted and actual frame capture time is 10ms. Furthermore, video stream delay decreased from 200ms to 115ms on average.**

I. Introduction

In the field of CCTV technology, a major issue is the lack of precise timestamps in video streams. Most standard CCTV cameras don't include this feature, which is crucial for tasks like real-time computer vision. This gap makes it difficult to merge data from different sensors effectively. By default, the cameras had large buffers: the delay increases over time; while the camera is reloading, the CCTV system is not operational. The picture sometimes had chaotic brightness spikes, affecting the efficiency and accuracy of the computer vision solutions. Also, since most camera software is closed-source, making custom changes is challenging. Our paper presents a method to upgrade CCTV cameras for accurate timestamping. We analyze the camera hardware, set up a connection, install OpenIPC OS, and then read images directly from the camera's sensor. Our key advancement is adding NTP-based timestamps to the video stream, which can then be shared using protocols like RTSP. This approach greatly enhances CCTV systems for various real-time applications.

II. Hardware Setup & OpenIPC Installation

This firmware can be used for various cameras, including, but not limited to HiSilicon chips [1]. However, we provide the specifications of the setup used in this research for simplicity of reproduction:
Matrix: 1/2" Approx. 2.13M-Effective Pixel Color CMOS Image Sensor IMX385
CPU: HikVision Hisilicon Hi3516D
108 MB RAM; 16 MB Storage
The installation of OpenIPC can be done via the UART protocol through the U-boot menu. U-boot is a section that, in addition to daily loading and launching the kernel, can be used by us to update the system over the network or MMC (SD card) and to save backups of both the entire flash drive with firmware and particular parts. To get to the U-boot menu, you need to have a USB UART adapter, find inconspicuous RX, TX contacts on the camera board, connect to them using special clothespins or solder, and run a terminal program like PuTTY for Windows, minicom for Linux or SerialTools for MacOS (as shown on Figure 1). Once connected, you should be able to Ctrl+C after replugging the camera to access U-boot (as shown in Figure 2).

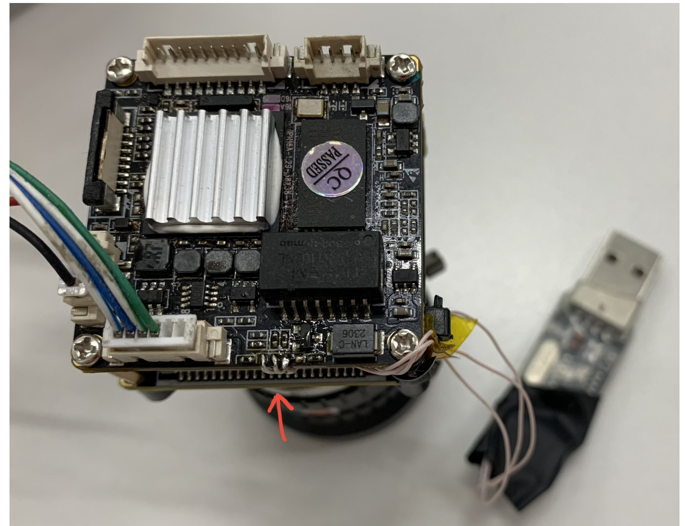

Fig. 1. Locate UART pins

We recommend backing up the original firmware before proceeding with the installation of the OpenIPC OS. Follow the OpenIPC provided installation instructions for various chipsets [1].

III. Matrix driver modification

While OpenIPC supports many combinations of processors and camera matrices, in our case, support for an IMX385 matrix was not implemented by the developers.

```
hisilicon # help
?       - alias for ,help'
base    - print or set address offset
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
cmp     - memory compare
cp      - memory copy
crc32   - checksum calculation
ddr     - ddr training function
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
getinfo - print hardware information
go      - start application at address ,addr'
help    - print command description/usage
loadb   - load binary file over serial line (kermit mode)
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mtest   - simple RAM read/write test
mw      - memory write (fill)
nand    - NAND sub-system
nboot   - boot from NAND device
nm      - memory modify (constant address)
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sf      - SPI flash sub-system
tftp    - tftp - download or upload image via network using TFTP protocol
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor version
```

Fig. 2. U-boot help

First of all, matrix configuration happens via I2C or SPI. In order to be able to communicate correctly with the matrix, you need to change the configuration of the pins by studying the datasheet and setting the flags in the corresponding registers.

The next important part is the matrix setup driver. Usually, it is built into the kernel or compiled into the streamer; however, OpenIPC implemented a better approach – loading it as a dynamic library. After debugging the camera from the original firmware, it was found that the driver is compiled into the kernel, and it is problematic to get it from there. Several drivers in dynamic library format were also found on the Internet, and the driver source code for a similar matrix. After cross-compilation, as well as modifications, it was possible to get a partial picture from the camera (with color problems). However, attempts to compare the state of the ISP registers and matrix registers with the original firmware did not give any clues as to what the problem was.

One of the drivers found on the Internet had an i2c connection error, however, according to a preliminary analysis (decompilation with Ghidra software [6]), it was recognized as theoretically suitable. After manually setting the registers of the matrix, a correct and clear picture was obtained without artifacts. Therefore, a patch was implemented in the form of our own dynamic library and its loading using LD_PRELOAD. In this way, it was possible to implement the correct communication of this driver via i2c with the camera matrix.

Another important part of this driver is the ISP (image signal processor) settings and setup for correct operation with the camera. It was this part that caused artifacts in the image, and it was correct in the found driver. The settings are very voluminous and detailed, and any inconsistency with the matrix datasheet has an extremely negative effect on the operation of the ISP system and the resulting stream.

IV. Custom Streamer

Default streamer OpenIPC Majestic [9] – has a closed source code, that's why it was not possible to use it to customize its code for the timestamp addition.

However, OpenIPC has several poorly supported open-source streamers, including OpenIPC Mini Streamer [7] and OpenIPC SmolRtsp [8]. OpenIPC Mini Streamer is only supported for the latest processors, and OpenIPC SmolRtsp has no bindings and ISP/VI setup, i.e., it implements purely frame transfer. That's why Mini didn't work out of the box on our camera due to a mismatch of HISI_OSDRV – a set of system internals, ISP system, and camera streaming tools built into the kernel – and the actual streamer realization.

A long way of modifying and setting up the streamers to work correctly with the internal streaming system and h264 encoding was covered. As a result, a hybrid streamer of the two above-mentioned streamers was obtained, which worked stably and did not have the problems of the original firmware streamer – correct and maximally small buffers were implemented, and the time from receiving the encoded frame from within the system to its actual sending to clients was minimized.

Several ISP optimization setups have been made to minimize latency within the ISP.

V. Timestamps

The most popular approach for absolute timestamping in the RTSP video stream is to send RTCP SR packets, which contain the mapping between relative RTP timestamps and absolute NTP timestamps [3]. However, the RTCP protocol is not implemented by smolrtsp or mini streamer at all. Therefore, two other modern approaches are used in our custom streamer. The first one embeds a timestamp into each frame using a SEI NALU [2] within the H264 encoder. The second embeds a 64-bit NTP-format timestamp into RTP packets.

A. H264-based timestamping

H264-based timestamping ensures that the first timestamp is embedded in each frame at the earliest stage of the pipeline. This is done by injecting it into each frame using a SEI NALU [2]. Furthermore, the second timestamp is appended when transferring frames from the internal OS buffer to the streamer's buffer and is later used to reduce the synchronization error in cases where processing is delayed.

B. RTP data packets

The second method of timestamp injection involves using 64-bit NTP Header Extension [4] that was proposed in November 2010 and yet was implemented by GStreamer only in the recent 1.22 version. The structure of an RTP packet with this extension (See Figure 5) stores the NTP timestamp as a 64-bit value where the higher 32-bit part represents the number

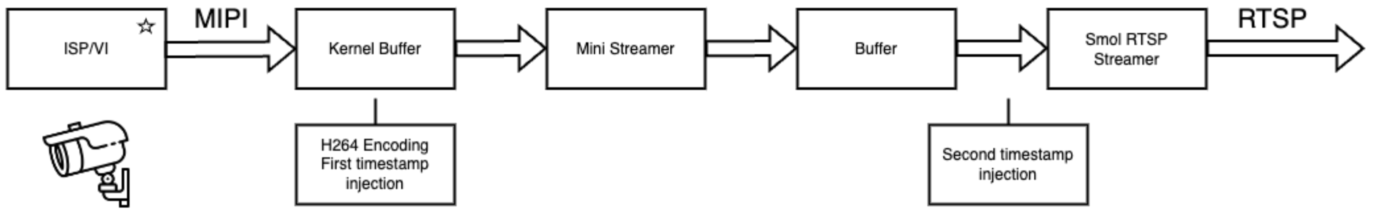

Fig. 3. Streamer's architecture

Fig. 4. ISP Finite State Machine

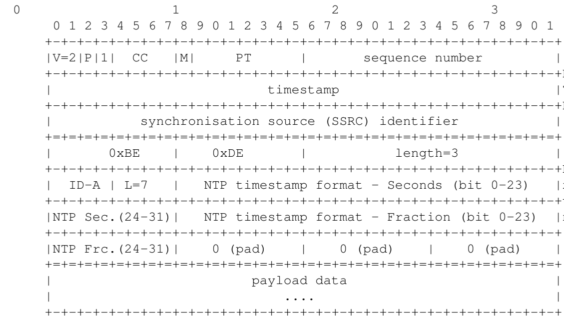

Fig. 5. RTP packet with 64-Bit NTP Header Extension

of seconds, and the lower 32-bit part represents the fraction. The injected timestamp is calculated as the average of the first H264-based timestamp and the second H264-based timestamp.

C. Timestamp retrieval on the client side

The GStreamer pipeline is used for image retrieval. As mentioned earlier, in the recent 1.22 version of GStreamer, the rtspsrc element is able to automatically add GstReferenceTimestampMeta [5] to the frame buffer if the received RTP packet has a 64-bit NTP extension. In fact, that is the only extension that GStreamer handles directly. At the end of the GStreamer pipeline, the processed frame buffer is pulled from the appsink element and is later displayed on the screen with timestamp overlay, which is stored in the buffer metadata.

VI. Debugging Tools

Utilize the ipctool [1] for direct camera configuration and testing. If the matrix is not detected on the i2c interfaces, investigate the hardware connections. Going further, install ipctool on a default camera firmware to copy the registers. During debugging, you need to have processor registers in front of you, camera registers available via i2c, and a full proc dump of the original camera in different operation modes.

For drivers, packages debugging (in particular, to debug and extract useful setup features for ISP of the closed source majestic streamer), the following tools can be used: Ghidra [6], cross-compiled gdb for the camera as a server, patching using LD_PRELOAD.

Since the amount of permanent memory on the camera is very small, and all programs are minimized and simplified, it is easy to extract useful information at the assembler level.

VII. Results

We successfully eliminated the need for periodic camera reloads by optimizing buffer management, thereby reducing delays and ensuring continuous operation. Additionally, we resolved the issue of random white flashes in the video feed, leading to a more stable and reliable image output. The table I illustrates the enhanced synchronization precision achieved with our solution compared to the initial firmware. The average delay time significantly decreased from 200 milliseconds in the initial firmware to 115 milliseconds in our solution. Furthermore, the error metric shows a marked improvement. While the initial firmware exhibited a cumulative error, our solution achieved a mean error of 0 milliseconds with a standard deviation of 10 milliseconds.

TABLE I
COMPARISON OF SYNCHRONISATION PRECISION

Firmware Version	Average Delay Time (ms)	Error (ms)
Initial Firmware	200	Cumulative Error
Our Solution	115	mean=0ms, std=10ms

VIII. FUTURE WORK

We plan to delve into the potential of accessing and modifying the camera's kernel code. This advanced level of customization would enable precise control over the timestamping process. Specifically, we aim to implement ”same frame” timestamp injections, ensuring each frame carries a timestamp that accurately reflects the moment of its capture. This approach promises to provide the best accuracy utilizing all the potential NTP clocks can provide.

REFERENCES

[1] “OpenIPC: supported chips and installation manual” https://openipc.org/supported-hardware/featured
[2] “H.264 : Advanced video coding for generic audiovisual services”, https://www.itu.int/rec/T-REC-H.264-202108-I/en
[3] “RTP: A Transport Protocol for Real-Time Applications”, https://datatracker.ietf.org/doc/html/rfc3550#section-6.4.1
[4] “Rapid Synchronisation of RTP Flows”, https://datatracker.ietf.org/doc/html/rfc6051
[5] “GStreamer rtspsrc documentation”, https://gstreamer.freedesktop.org/documentation/rtsp/rtspsrc.html?gi-language=c#rtspsrc:add-reference-timestamp-meta
[6] “Ghidra Software” https://github.com/NationalSecurityAgency/ghidra
[7] “Mini Streamer” https://github.com/OpenIPC/mini
[8] “Smolrtsp Streamer” https://github.com/OpenIPC/smolrtsp
[9] “Majestic Streamer” https://github.com/OpenIPC/majestic