`UNV_v2.0`: Content Centric Live Video Streaming

Grigorios Stathis, George Smart, Obada Sawalha, Lorenzo Levrini, Hans Balgobin, Stelios Vitorakis and Yiannis Andreopoulos

University College London

Abstract[1]: We present `UNV_v2.0`, an open-source video streaming framework for the Linux operating system, based on the video for Linux (`v4l2` API) and the `FFmpeg` library. The purpose of the 2nd version of UNV is low-delay live capturing and streaming over IP, with a choice of codecs and protocols to use. The architecture and main features of the software are presented and bandwidth utilization, subjective quality assessment and delay results are given for a video conferencing application and a surveillance application scenario over Wi-Fi and over a campus eduroam network.

1. Introduction

Low-delay video streaming is now a prevalent service for entertainment, business, surveillance and social media enterprises [1][2]. Within the last year alone, Google, Microsoft, Facebook, as well as several major Internet Service Providers around the world have focused on product offerings including low-delay video capture and streaming over IP for conversational or "rich media" applications [1]. The upcoming HTML5 standard with the foreseen "video" element will include several APIs supporting video capture, encoding and decoding directly through the web browser [1], fuelled by the *de-facto* presence of cheap webcam hardware on all laptop computers, tablets and smartphones today.

In this paper we present `UNV_v2.0`, a software solution for low-delay video capture, coding and streaming via the Linux operating system. `UNV_v2.0` offers:

- an open-source solution for video capture via the `v4l2` (video for Linux) API of the Linux kernel, and encoding&streaming via a C/C++ implementation that is easy to extend, as it is entirely based on the `FFmpeg` library [3]; our solution allows both offline and live content to be streamed;
- video playback at the client through VLC[2] or indeed any other player supporting raw (unwrapped) H.264/AVC or MJPEG streams[2];
- support for streaming via multiple ports for content-driven or traffic-driven prioritization [4];
- optional use of MKV wrapping for multi-stream rich-media transmission over IP;
- efficient resource usage that enables its operation even in low-end netbook computers.

We created `UNV_v2.0` as today there is no open-source live video streaming that functions with new-generation coding schemes (e.g. H.264/AVC) and also provides a simple implementation that operates with the mainstream `FFmpeg` library. Mainstream solutions such as Ekiga or Skype are either proprietary (closed-source) software, or utilize customized players and encoding tools, thus making further application development based on them difficult to maintain and support. Other open-source tools, such as Gnome Empathy or even direct VLC-to-VLC streaming, have significant implementation complexity due to the bundling of many services and protocols into a single project, thus making further application development in their codebase a difficult task. On the contrary, all I/O routines and coding/decoding in `UNV_v2.0` is handled very efficiently by using "off-the-shelf" components from the `v4l2` API and the `FFmpeg` library; both of these software packages are actively maintained by a large community of developers; thus one can concentrate on the streaming session development and on the offering of content-centric functionalities to the user (e.g. video analysis, content-adaptive streaming, etc. [4]) while utilizing the existing codebase of these projects.

2. `UNV_v2.0` Architecture and Main Features

The project follows the conventional client-server architecture, outlined in Figure 1. Possibilities for wrapped bitstream transmission from the server to the client exist in `UNV_v2.0` via the support of the

[1] **Acronym synopsis:** UNV: UNV, Not VLC; VLC: VideoLAN Client; IP: Internet Protocol; HTML: Hypertext Markup Language; MKV: Matroska Video; RTSP: Real-time Streaming Protocol; TCP: Transmission Control Protocol; UDP: User Datagram Protocol; Wi-Fi: wireless systems certified by the Wi-Fi Alliance; AVC/H.264: ISO-IEC Advanced Video Coder Standard/ITU H.264 Recommendation; MJPEG: Motion JPEG codec; RAM: Random Access Memory; RTP: Real-time Transport Protocol; RTT: Round Trip Time.

[2] http://www.videolan.org/, http://wiki.videolan.org/Codec

MKV wrapper [4]. The client is using the VLC player for efficient real-time video decoding and playback. In addition, VLC can provide the current playback point (F_o) and the client can periodically send it to the server for rate and transmission adaptation, or adaptation of the server (local) capturing play-out point (F_l) for soft real-time content streaming (i.e. end-to-end delay of a few seconds). The network operations indicated in Figure 1 involve packet fragmentation, usage of RTP encapsulation and transmission via UDP or TCP to the client's designated IP address and port(s). Further details on these operations are included below. The multi-port streaming potential of UNV is detailed in [4].

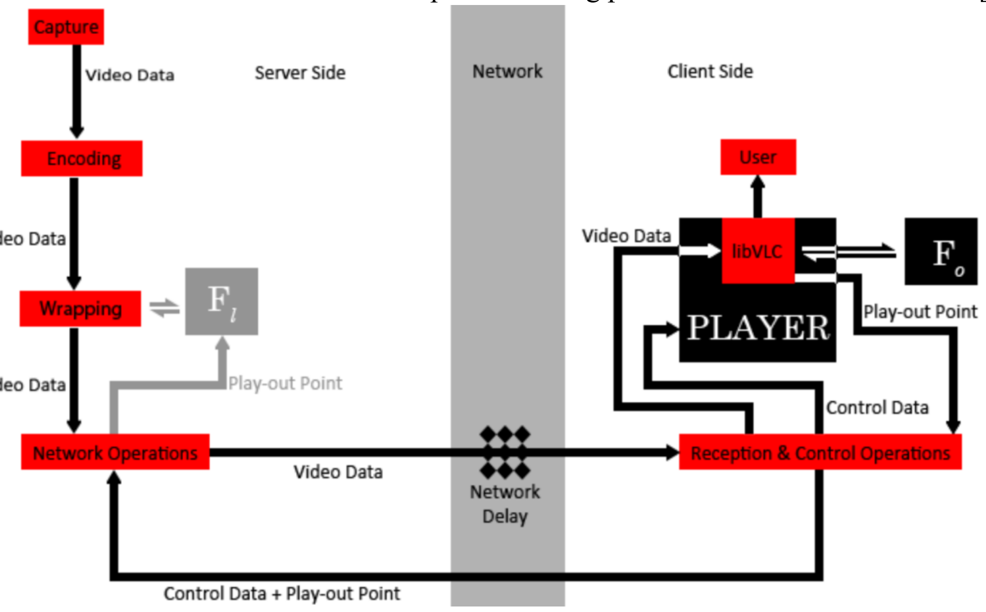
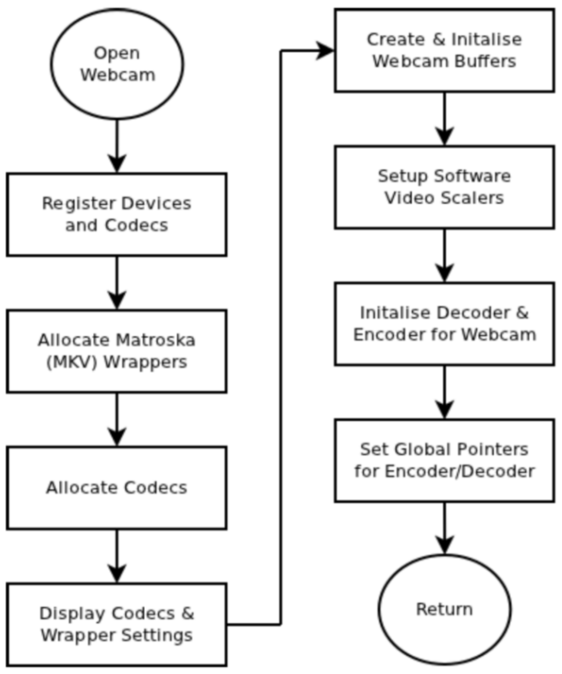


Figure 1. Block diagram outlining the `UNV_v2.0` architecture.



Open Webcam
Register Devices and Codecs
Allocate Matroska (MKV) Wrappers
Allocate Codecs
Display Codecs & Wrapper Settings
Create & Initalise Webcam Buffers
Setup Software Video Scalers
Initalise Decoder & Encoder for Webcam
Set Global Pointers for Encoder/Decoder
Return

AVPacket	Any binary data except raw frames is treated as an AVPacket.
AVFrame	FFmpeg's "raw" format for video. Webcam AVPackets are decoded to AVFrame before encoding.
AVCodecContext	Contains all parameters used for a given codec. It is initialised to default values using an FFmpeg method before populating with desired parameters.
AVFormatContext	Contains all parameters associated with a wrapper format, in this case MKV. Passed as a pointer to the wrapping method, and contains the buffers sent over the network by UNV.
AVCodec	Contains all codec information and settings.
AVStream	Contains stream information. It has an AVCodec associated with it.

Figure 3. Key `FFmpeg` structures used by the `UNV_v2.0` server.

Figure 2. Flowchart for the webcam opening function.

The server parses the command-line parameters and then, if live video streaming of webcam content is required, opens the webcam after establishing connection with the client (see Figure 2 for the flow-chart of the related function). First, the video header is transmitted, which includes the MKV header if MKV wrapping is used. Subsequently, under "live webcam" operation, once a video frame is captured from the webcam via the v4l2 API, it is encoded via the FFmpeg's x264 (open-source H.264/AVC implementation) or MJPEG coder and the produced video packet is fragmented into pieces of a predefined maximum size, which are then transmitted to the client via a UDP or TCP connection. Fragmentation of video packets prior to transmission is necessary in order to avoid excessively large packets, especially for the case of the MJPEG coder and the intra frames of the H.264/AVC.

The implementation of the frame capturing, encoding and network-related processing is done on separate threads for increased efficiency in modern multicore processor environments. A summary of the key FFmpeg structures used at the server is given in Figure 3.

The client communicates with the server through an RTSP/TCP control session with the standard control commands ("SETUP", "PLAY", "STOP", etc.). Furthermore, the client launches VLC (selected because of its excellent error-recovery capabilities) and passes each received video packet (after assembling all its fragments) to the standard input of VLC via an operating system pipe. The simple implementation via a system pipe makes the client independent of the video player code and allows for a simple design. The current playout point can be retrieved via libVLC and sent back to the player periodically via the RTSP/TCP session. VLC is launched with only 10ms buffering and zero jitter (`--file-caching 10 --clock-jitter=0`) in order to allow for low-delay decoding and playback. Real-time flow-control mechanisms can be implemented via the knowledge of the playback point and the control of the passing of received video packets from the client buffer to VLC.

3. Experimental Results

We focus on two scenarios: *(i)* a typical "video meeting" application, where each end launches a client and a server and real-time video streams are captured, encoded and streamed bi-directionally; *(ii)* a "video surveillance" application where a camera monitors an urban area and transmits compressed video to a remote location via a UDP or TCP connection.

For each case, we performed two trials: one via a stand-alone Wi-Fi router, and one via the University College London's campus eduroam wireless network. A low-end laptop (AMD V120 2.2GHz, 3GB RAM) and a netbook computer (Intel ATOM 1,66GHz, 1GB RAM) were used for our experiments (both running Kubuntu Linux). For video encoding, MJPEG is used with fixed quantization (qmin=qmax=90) and H.264/AVC is used with the constant quality encoding (crf=32 in the x264 encoder), and with only one reference frame and only P frames for low-delay encoding and decoding. The video resolution was set to 640x480 pixels and 24 frames per second were used. End-to-end delay is measured very accurately by using a millisecond clock at the client side and capturing the client monitor with the server webcam; thus, the client shows simultaneously *its native time as well as what the server captures at that instant* (see the example given in Figure 4). This simple measurement technique is accurate and applicable to any live capture and streaming framework.

The obtained delay results are given in Table 1, where we also present the average delay of Skype and direct VLC-to-VLC streaming over eduroam. Our UDP solution achieved average end-to-end delay of 402ms on Wi-Fi and 567ms on eduroam, which is still 2.4~3.4 times higher than that of Skype but it is an order of magnitude smaller than that achieved by VLC-to-VLC streaming. Packet RTT was found to be around 15ms on average and the server capturing and encoding delay was measured to be around 15ms. Hence, the usage of VLC as an external component at the client comes with a significant delay penalty in the current implementation. Thus, future work should attempt further delay reductions at the client, ideally without sacrificing the separation between client and the player.

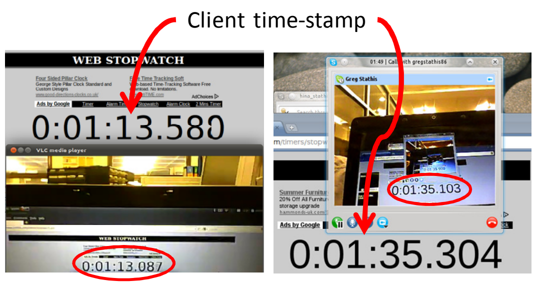


Figure 4. End-to-end delay measurement. Encircled: Time stamp contained in the received stream from the server at that instant in UNV_v2.0 (left) and Skype (right).

Wi-Fi, UDP, H.264	402
Wi-Fi, TCP, H.264	636
Wi-Fi, UDP, MJPEG	1105
Wi-Fi, TCP, MJPEG	17193
eduroam, UDP, H.264	567
eduroam, TCP, H.264	667
eduroam, UDP, MJPEG	832
eduroam, TCP, MJPEG	9569
eduroam, Skype	168
Wi-Fi, UDP, VLC-to-VLC	4669

Table 1. Delay measurements (in ms, averaged over multiple trials).

Concerning the obtained video quality (Figure 5), H.264/AVC achieved higher visual quality than MJPEG for all our experiments, despite our attempts to set quantization parameters leading to similar visual quality. However, UDP transmission led to occasional frame drops that were significantly more visible with H.264/AVC than with MJPEG due to the use of motion compensated prediction.

Finally, concerning bandwidth utilization, Figure 6 shows traffic profiles for indicative experiments with both codecs over eduroam, under TCP and UDP. Ignoring the initial traffic peak due to stream initialization, UDP transmission led to a smooth traffic profile (variations between 50kbps~100kbps) with smooth playback; on the contrary, TCP suffered from very high traffic variability (that also led to somewhat irregular video playback at the client) due to the accumulation of multiple packets in the server buffers (due to delayed packet acknowledgments and TCP's backoff policies). In all our experiments, despite offering lower quality, MJPEG led to significantly higher bandwidth requirements and comparable or higher variations than H.264/AVC.

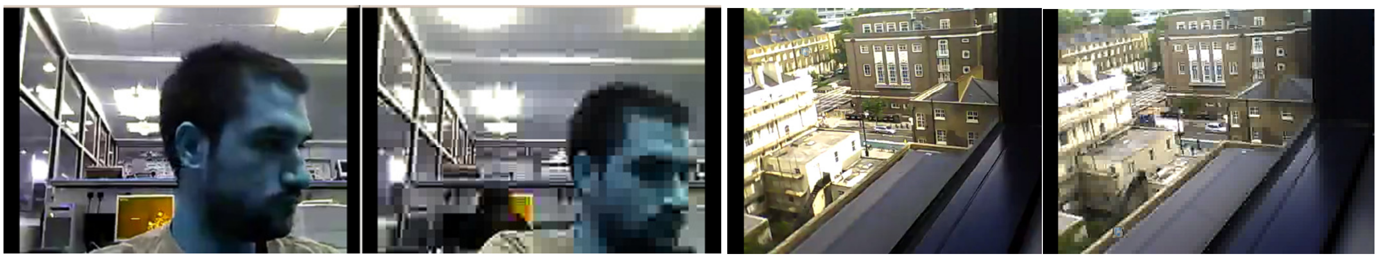


Figure 5. Visual comparisons of H.264/AVC encoded video (left halves) and MJPEG encoded video (right halves) for the conversational (left) and surveillance applications (right).

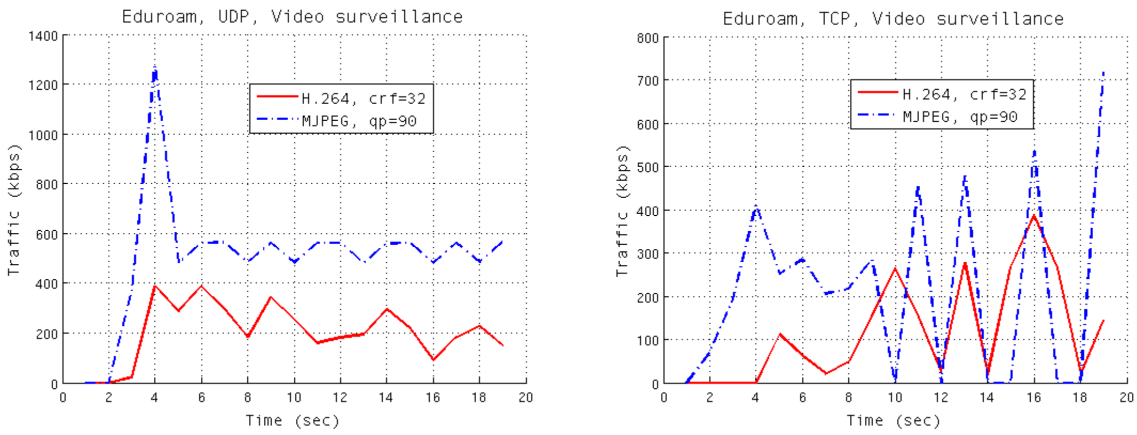


Figure 6. `UNV_v2.0` traffic (in kbps, averaged from the arrival packet traces) of H.264/AVC and MJPEG streams under eduroam via: UDP transmission (left) and TCP transmission (right).

4. Conclusion

Based on the `v4l2` API and the `FFmpeg` library, the proposed `UNV_v2.0` software solution (available at http://www.ee.ucl.ac.uk/~iandreop/UNV.html) provides an open-source framework that allows for low-delay video capturing, encoding and streaming. While still in development stage, our solution provides multiple codec support, as well as UDP and TCP streaming, with a simple software design that allows for the use of any state-of-the-art video playback framework (such as VideoLAN's VLC) supporting the utilized codecs. Even though `UNV_v2.0` is already significantly more delay-efficient than direct VLC-to-VLC streaming, further improvements can focus on the client side in order to bring the end-to-end delay from 400~570ms down to the 150ms range achieved by commercial offerings for real-time conversational services over IP.

References

[1] E. Ackerman and E. Guizzo, "5 technologies that will shape the web," *IEEE Spectrum*, vol. 48, no. 6, pp. 40-45, June 2011.

[2] A. Roy, N. Gale and L. Hong, "Automated traffic surveillance using fusion of Doppler radar and video information," *Mathematical and Computer Modelling*, vol. 54, no. 1-2, pp. 531-543, July 2011.

[3] S. Tomar, "Converting video formats with FFmpeg," *Linux Journal*, no. 146, June 2006.

[4] I. Tsakos and Y. Andreopoulos "Multiport streaming of Matroska (MKV) video over IP", *Proc. London Communications Symposium*, paper 35, Sept. 2010.