Low-latency Live Video Streaming over Low-Capacity Networks

Giovanni Gualdi, Rita Cucchiara
Dipartimento di Ingegneria dell'Informazione
University of Modena and Reggio Emilia
Via Vignolese, 905/b - 41100 Modena, Italy
{gualdi.giovanni, cucchiara.rita}@unimore.it

Andrea Prati
Dipartimento di Scienze e Metodi dell'Ingegneria
University of Modena and Reggio Emilia
Via Amendola, 2 - 42100 Reggio Emilia, Italy
prati.andrea@unimore.it

Abstract

This paper presents an effective system for streaming over low-capacity networks (such as GPRS and EGPRS) of live videos with low latency. Existing solutions are either too complex or not suitable to our scope. For this reason, we developed a complete, ready-to-use streaming system based on H.264/AVC codec and UDP/IP stack. The system employs adaptive controls to achieve the best trade-off between low latency and good video fluency, by keeping the UDP buffer occupancy at the decoder side between two given levels. Our experiments demonstrate that this system is able to transmit live videos at CIF format and 10 fps over GPRS/EGPRS with very low latency (1.73 sec on average, basically due to the network delay), good fluency and average quality, measured with PSNR, of 31 dB on GPRS at 23 kbps at 10 fps.

1. Introduction

Ubiquitous multimedia access (UMA) is one of the hottest topics of the last years in the multimedia scientific community. Providing access to multimedia data, in particular videos, from everywhere by means of mobile devices (such as PDAs or last-generation cellular phones) is becoming a must for commercial telecom providers. Possible applications of such a technology include home-consumer entertainment and digital TV broadcasting, video conferencing, telemedicine and telemanipulation, military applications, and remote video surveillance.

All these applications, and in general UMA scenario, contain several technological challenges. First, videos are complex data, containing a wide variety of information and very rich content, but also posing serious problems in terms of amount of data to be transferred on the network and of computational resources. Moreover, mobile devices and UMA scenario require accessibility through wireless networks, either 802.11 WiFi, 3G networks such as UMTS (Universal Mobile Telecommunications Service), or even 2G or 2.5G networks such as GPRS (General Packet Radio Service).

The target application of this work belongs to the broad class of remote video surveillance. The scenario is that of a camera mounted on a moving vehicle that transmits *live* videos to a fixed receiver. The possible application here is the remote patrolling and monitoring of areas by means of robots. In addition, the moving transmitter can be geographically very distant from the receiver and can move almost everywhere.

Therefore, in addition to the above-mentioned general challenges, our case has the following requirements:

1. *live video delivery* requires efficient video coding and transmission, and does not allow constant bitrate (CBR) coding; off-line optimizations are not possible;
2. since the transmitter can move in everyplace, a *network with* an (almost) *ubiquitous territorial coverage* is necessary; therefore, WiFi or UMTS can not be currently used due to their limited coverage; thanks to its coverage of Italian territory (almost 100%), GPRS network has been selected as transportation layer; in particular, the GPRS-EDGE (Enhanced Data rates for GSM Evolution), also known as EGPRS, version has been used, given that it has the same coverage of GPRS but with a higher available bandwidth (theoretically between 160 and 236.8 kbit/s);
3. since the transmitter is moving, the wireless network is required basically on the *uplink of the transmission*; since the GPRS is an asymmetric protocol (giving more bandwidth in download than in upload), this further reduces the available bandwidth;
4. since video surveillance requires a *good understanding of the scene*, received live video needs to be of both high quality and good fluency of images;
5. since video surveillance requires often *quick reaction to changes* in the scene (in order to allow interaction

with the remotely observed scene), a low latency in the video playback is required.

The first requirement calls for video streaming since delivery of live videos does not permit storage and off-line compression. Then, being restricted to use low-capacity networks like GPRS-EDGE in upload (requirements #2 and #3) means that standard, commercial compression techniques, such as MPEG-2 and MPEG-4 are not sufficient. High resolution (at least CIF - 352x288 - or QVGA - 320x240) videos on a bandwidth of maximum 236.8 kbps are achievable with MPEG-4 only by reducing either the frame rate (affecting video fluency) or the image quality, and both these drawbacks will be in contrast with requirement #4. High interest is shown in recent years on the evolution of MPEG-4 and H.263, the H.264/AVC [1] compression technology, that offers extremely good video quality even at very high compression rates: this makes it very suitable for the scope of our project. As a drawback it presents higher computational complexity if compared to MPEG-2 and MPEG-4 (both on encoder and decoder side), though it is currently possible to use it on real-time systems if the encoder is fed with reasonable size video streams and properly tuned [17].

Existing commercial or off-the-shelf streaming solutions are not sufficient to meet our requirements. For example, Microsoft Windows Media® suite (Encoder, Streaming Server, Player) [5] is a complete and powerful tool for video streaming working with both stored and live videos. Unfortunately, since it has been developed more for entertainment purposes than for real-time live video streaming, this software introduces high latency (see section 4), that does not meet our requirement #5. Moreover, this suite is pretty strict in the selection of the video codecs, basically providing the Windows proprietary codecs only (Windows Media Codecs). Other tools such as Helix Streaming Server® [6] or Darwin Streaming Server® [2] have been devised to handle multiple connections and severe computational load, more than to be used for low-latency live video streaming.

A separate discussion is deserved to Videolan [7]. Videolan (VLC) is a open-source tool that provides many compression standards (H.264 among them), and streaming technologies. Differently from the previous tools, Videolan has been designed for research and not with commercial purposes. For this reason, the interface is very limited, but the efficiency and flexibility is much higher than the above-mentioned systems, and the high portability makes it even more interesting. In spite of these last comments, Videolan has shown limitations that conflict with the requirements of our project: at first, real-time H.264 video streaming has a latency of more than 3.5 seconds on average with standard operational conditions (see section 4). Moreover, H.264 streaming is allowed only on MPEG TS (Transport Stream) encapsulation: [16] demonstrate that this is a drawback, since using the stack MPEG-TS/UDP/IP introduces more than the double of overhead than using the stack RTP/UDP/IP. Finally, Videolan on H.264 is pretty weak in the bitrate control and does not give access to the full palette of parameters of the H.264 implementation (based on X.264 [3]).

For all the above reasons, we decided to develop our own live video streaming system, based on H.264/AVC [1, 20, 17] and UDP as transportation layer protocol, specifically devoted to work on low-capacity networks and to meet the above-reported requirements by means of several improvements to make the streaming adaptive.

The paper structure is as follows. Next section will report on the existing solutions for live video streaming. The system will be described in section 3 and its performance analyzed in section 4. Eventually, section 5 will contain our conclusions and suggestions for future directions.

2. Related works

Video streaming has reached its peak of interest in the scientific community in the last ten years. A survey paper on video streaming has been reported by Lu in 2000 [14]. It reports of and discuss about the relevant signal processing issues related to video streaming and proposed possible solutions.

In general, regarding video streaming, researchers have addressed the problem from several different perspectives. For instance, Lu *et al.*in [15] basically considered the allocation of computational resources for a server that encodes live videos and receives more clients with different priorities in accessing to the videos. Instead, works such as [10] and [11] mainly addressed video streaming in terms of system architecture. In particular, Conklin *et al.*in [10] proposed several models of communications (based on RealNetworks® products) for video delivery on the Internet, not considering low-capacity networks. Guo *et al.*in [11] proposed a very nice application for live video streaming between moving vehicles and depicted an interesting architecture for data synchronization. However, their system is based on 802.11 WiFi networks and thus it is not suitable to our case.

Some previous works have proposed systems for video streaming over low-capacity networks, such as GPRS. For instance, Lim *et al.*in [12] introduced a PDA-based live video streaming system on GPRS network. The system is based on MPEG-4 compression and contains several computational optimizations for working on a PDA. It can achieve a frame rate of 21 fps (frames per second) at the encoder side and 29 fps at the decoder side for transmitting at 128 kbps a video in QCIF (176x144) format. However, their system drops to 2-3 fps when transmission is over GPRS. This limitation is basically due to the use of MPEG-

4. Moreover, no information on the latency of the system is provided. The work in [13], instead, solves the problem of transmitting real-time videos over GPRS by using frame skipping. Unfortunately, in our target application, frame skipping is not suitable, since crucial information can be missed.

The recent delivery of the new video compression standard called H.264/AVC [1] opens to new possibilities in video streaming. In fact, the primary goals of this standard are *improved video coding* and *improved network adaptation* [19]. Therefore, many features make it suitable for wireless video transmission, such as improved error resilience, excellent bitrate adaptation, buffering, and so on [18]. Moreover, the standard distinguishes between two conceptual layers, called *Video Coding Layer* (VCL) and *Network Abstraction Layer* (NAL). In particular, the NAL provides the interface between the video codec and the outside world and is designed to be compliant to many different networks and transportation layers (RTP, UDP, TCP, etc.).

An interesting implementation of H.264/AVC for video streaming has been proposed by Antonios Argyriou in [9, 8], where the primary contribution relies on the introduction of a new transport layer protocol called Stream Control Transmission Protocol (SCTP). Due to the message oriented nature of this protocol, it is particularly suitable for handling multiple streams and multi-client access to videos. However, our implementation based on the stack UDP/IP (see next section) provides us with a efficient tool able to meet all the requirements reported in section 1. In fact, the potential benefits of SCTP are not necessary for our application, where a single video (with only video data and no audio) has to be transmitted to a single receiver, hence not requiring data multiplexing.

3. System description

The main objective of our system is to *deliver live video streams over GPRS/EGPRS networks with low latency and good video fluency*. Since we are interested in sending video streams with good quality and high frame rate, we selected H.264 in its open-source X.264 [3] implementation as video compression algorithm. Moreover, since our system aims at delivering only video (image) data with not audio and on a point-to-point basis, we simply packetized the X.264 stream on UDP datagrams.

The architecture of the system is sketched in Fig. 1. The server and the client applications are multi-threaded, and each block of the schema represents a main task to be performed and is processed by one dedicated thread. For the ease of future implementations, we decided to develop a C#/MS Visual .NET application that incorporates native C/C++ modules. A video coming from either stored or live source is loaded by the server application which converts video data into a YUV (4:2:0) stream. This stream is converted in a H.264 stream by using the X.264 encoder. It should be mentioned that we modified the original X.264 source code by allowing it to load not only videos from file system, but also from a generic circular buffer, that allows higher flexibility, since it can be easily fed with any kind of source (file system, video device, video server, etc.). In our case the video source is obtained through a standard USB camera (Creative WebCam PRO): the dedicated video grabbing thread provides the YUV frames to the circular buffer. On the other side, the encoder thread asynchronously extracts them from the buffer. Having asynchronous threads optimizes the processing since the execution of each thread is basically independent by the other's. Moreover if the grabbing rate is higher than the encoding rate for short time, no video data will be lost. As drawback, the buffer introduces some latency; for this reason it is important to keep the buffer at low occupancy, therefore the grabbing frame rate should be set to be not higher than the average encoding frame rate. In standard conditions, with CU load not too high, this is a reasonable assumption.

The raw H.264 encoded stream is seamlessly forwarded to the network streamer that packetizes it into UDP datagrams of fixed byte size. UDP is preferable with respect to TCP due to its prioritized dispatch over the network, but this comes at the cost of possible loss of packets. Nevertheless, our tests have demonstrated that with our system over GPRS-EDGE or even GPRS is very robust since an extremely low rate of packet are lost or received out of order (see section 4).

At the receiver side, the client application extracts the UDP datagrams and merges them, rebuilding the H.264 stream. The H.264 decoder is based on `avcodec` and `avformat` libraries (from `ffmpeg` [4]), and converts the stream back into a RGB stream that can be displayed by the client application.

Even if video grabbing frame rate and playback frame rate will be set to the same values, the packet generation rate (encoder side) and the packet extraction rate (decoder side) might differs from time to time for several reasons, such as wireless network uncertainty, CPU overload either on encoder or decoder side, higher/lower CPU need (depending on the video complexity), and so on. Therefore, the UDP network buffer at the receiver side plays an essential role in order to reduce the effects of these discrepancies. The policy of use of the network buffer must be defined very carefully for two main reasons: firstly, the playback latency time is for obvious reasons, directly related to the occupancy of the buffer; secondly, if the packet generation rate remains higher than the packet extraction rates for long enough, the buffer might fill up and every packet received thereon would be lost. For these reasons, we implemented a simple adaptive algorithm to increase or decrease the buffer size dynam-

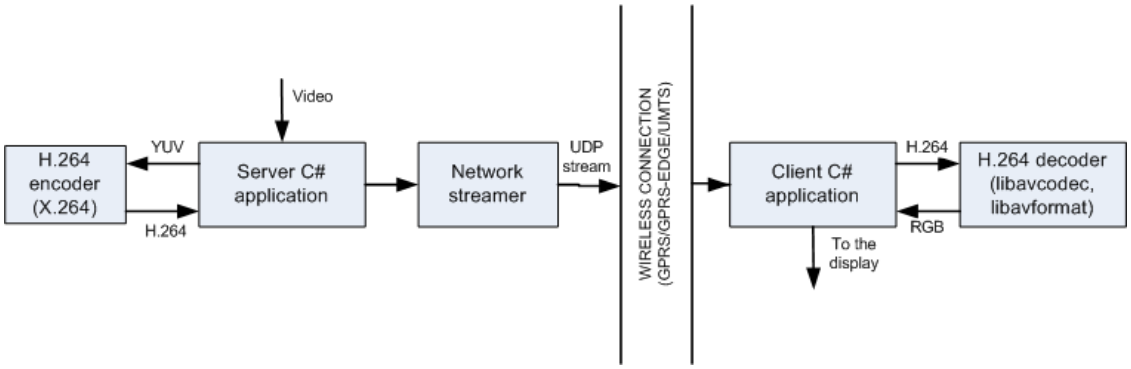


Figure 1. Architecture of the system.

ically: in practice, the algorithm either doubles the buffer size every time that it gets filled up beyond 80%, or halves it when its level decreases under 20%. These threshold values are computed empirically and depend on the network's conditions.

Both the above-reported points would push to keep the buffer as empty as possible, but this condition would affect the playback fluency, since the decoder would often had to wait on the network for incoming video data, resulting in the increase of the frame decoding time. Therefore, to achieve the best trade-off between low latency and good fluency, we implemented a simple algorithm to keep $B_{occ\%}$ between two values T_L and T_H (typical values are $T_L = 10\%$ and $T_H = 30\%$). The adaptation of the playback frame rate in function of the buffer occupancy is depicted in the scheme reported in Fig. 2. In practice, two values are monitored to implement the dynamic control: $B^t_{occ\%}$ that is the buffer occupancy in percentage (that needs to be kept between T_L and T_H) and $\Delta B^t_{occ\%}$ that is the discrete derivative of the buffer occupancy. In other words, if the buffer occupancy is high and continues to increase at a high rate (i.e., $\Delta B^t_{occ\%}$ is much greater than zero - condition 1 in Fig. 2), the frame rate of playback $FR^{t+1}_{playback}$ needs to be increased to compensate. Similarly, if the buffer occupancy is too low and it is still decreasing (condition 15), the playback frame rate needs to be reduced. The parameters α, β and γ satisfy the condition that $\alpha > \beta > \gamma > 1$ and typical values used in the system, obtained after several tests, are: $\alpha = 1.04$ (increase/decrease of 4%), $\beta = 1.01$ (1%) and $\gamma = 1.003$ (0.3%).

4. Experimental results

Evaluating the performance of a live video streaming system is not an easy task, since mostly based on the perceived quality of the live (not stored) videos by means of the human receiver. However, since our objective is to achieve both low latency and good video fluency, we evaluated them for our system.

It is really worth noting that the presented system is an effective tool for immediate use in our application. We have experimented with it for long-lasting trips by car, moving for more than 200 km and also at speeds higher than 120 km/h. The system works properly in all these conditions, even with the transmitter and the receiver hundreds of kilometers far.

4.1. Video latency

As stated more than once, low latency is a key feature of our application. At the very early stage of our research, we evaluated commercial tools for understanding whether they are suitable for us or not. We have compared Windows Media and Videolan with our system with a CBR transmission at 80 kbps on GPRS-EDGE network (with a bitrate tolerance of 0.1). Windows Media encoder has been set to reduce latency by not using the streaming server (only necessary to handle multiple client connections) and by removing all the buffers (without these adjustments the latency ranges from 10 to 15 seconds!). Videolan has been set to have the same X.264 parameters of our implementation, to use the stack TS/UDP/IP (the only available on UDP) and to skip late pictures.

	Windows Media	Videolan	Proposed
Mean (sec.)	4.15	4.07	1.73
Variance (sec.)	0.026	2.19	0.042

Table 1. Mean latency of Windows Media, Videolan and our system.

The comparative results are summarized in Table 1. It is straightforward to see the our system outperforms both the other tools, with an average latency of only 1.73 seconds, basically due to the network delay. Videolan performance is

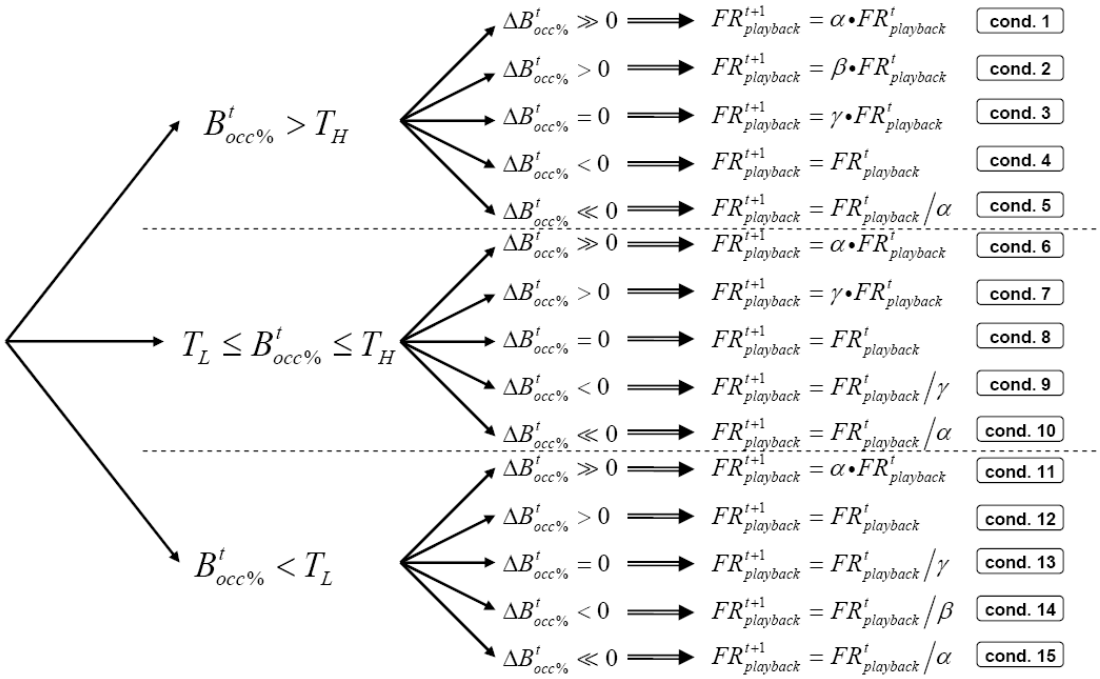


Figure 2. Explanation of the playback frame rate adaptation in function of the buffer occupancy ($\alpha > \beta > \gamma > 1$).

better than Windows Media, but much more unpredictable, with a very high variance.

It is also worth noting that our system achieves such a low latency with no packet loss. We both evaluated packet loss on GPRS and GPRS-EDGE network. In the latter, over 30 minutes of transmission from a car moving on the highway, about 10000 packets have been transmitted and all received in the correct order. The same test has been also performed on GPRS network, with 120 minutes of transmission from the car, with a total of about 15000 packets sent and only 10 (0.06%) lost (no out of order received). As a comparison, Videolan loses 5% of the packets on average (with many received out of order) also because the parameters have been set to reduce the latency by reducing the buffer sizes.

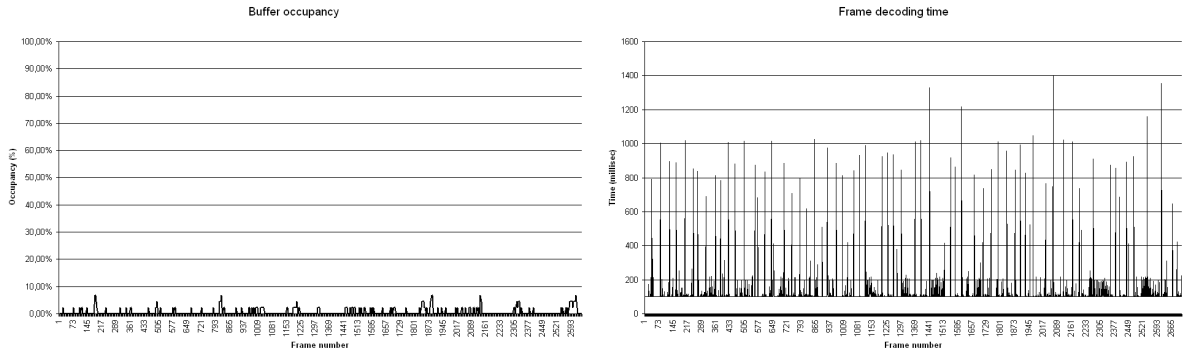
4.2. Video fluency

The opposite effect of reducing the latency can be to increase the frame decoding time at the client side, affecting the video playback fluency. For instance, the graph in Fig. 3(a) shows the buffer occupancy for a GPRS-EDGE transmission at 80 kbps with our system. By using the standard X.264 decoding (with the adaptive control described in section 3 - Fig. 2 - disabled), the buffer occupancy is always close to zero, resulting in an almost-ideal latency. However, as shown in Fig. 3(b), this will result in a frequent increase of the frame decoding time, from the normal 100 msec (due to a playback frame rate of 10 fps) up to 1400 msec (less than 1 fps). These continue changes in the frame decoding time will make the video played with poor fluency and this affects both the overall user satisfaction and the understanding of the scene.

Fig. 4 shows, instead, the result achieved by using our adaptive control. As described in section 3, when the buffer occupancy is lower than T_L (5% in this experiment), the control starts decreasing the decoding frame rate (see Fig. 4(a)) until the buffer occupancy is stable between T_L and T_H. This control reduces the unpredictability of the frame decoding time, and thus it improves the video fluency, as shown in Fig. 4(b). Obviously, the same behaviour can be also noticed in the case the buffer occupancy is higher than T_H, with the control increasing the playback frame rate to empty the buffer.

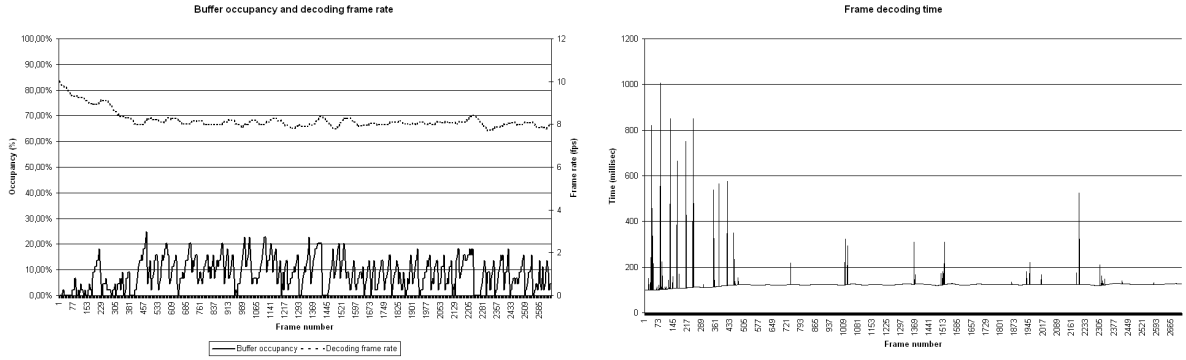
4.3. Video quality

As a final proof of the goodness of our system we evaluated the video quality transmitting a stored video of about 2500 CIF frames encoded at 10 fps, at different bitrates and over different networks. The graphs in Fig. 5 show the frame-by-frame PSNR (Peak Signal-to-Noise Ra-



(a) Buffer occupancy (%) (b) Frame decoding time

Figure 3. Buffer occupancy (in percentage) and frame decoding time (to demonstrate video fluency) in the case of standard X.264 decoding on a sample video



(a) Buffer occupancy (%) (b) Frame decoding time

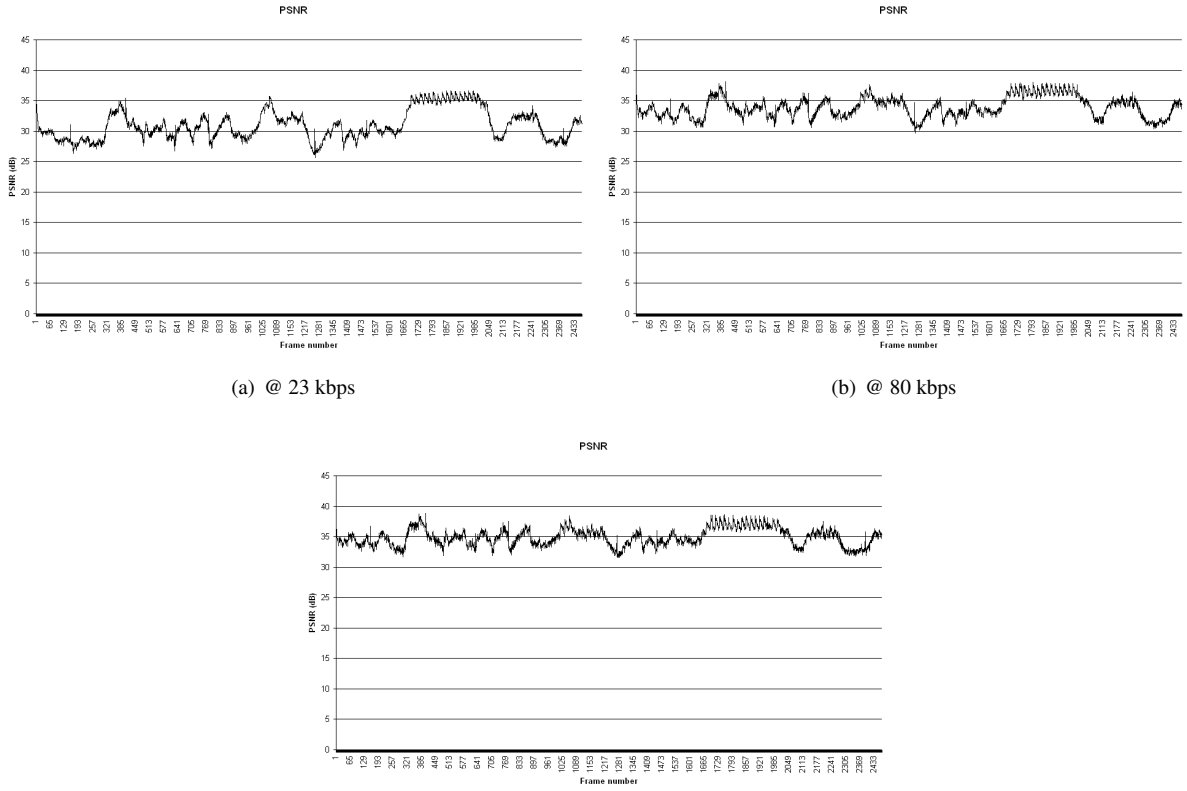
Figure 4. Buffer occupancy (in percentage) and frame decoding time (to demonstrate video fluency) in the case of our controlled-buffer (with $T_L = 5\%$ and $T_H = 25\%$) X.264 decoding on a sample video

tio) achieved in three of these experiments, with a bitrate of 23 kbps (resized at QCIF resolution) over GPRS networks (Fig. 5(a)), and with bitrates of 80kbps (Fig. 5(b)) and 120 kbps (Fig. 5(c)) over GPRS-EDGE. The difference in the PSNR along the video is due to the different situation present in the video (moving or stopped car). It is evident that, even with a very low bandwidth, the system achieves a quite good video quality, with an average PSNR of 31 dB.

5. Conclusions and future directions

In this paper, we described our effort in building a live video streaming system able to achieve both low latency and good video fluency. Our target application is the delivery of live video streams (with no audio) from a moving transmitter over GPRS/EGPRS networks (to have an ubiquitous coverage and no signal interruptions). With this objective, we examined the existing (commercial and non) solutions, coming up to the conclusion that none of them meets all the requirements of our application. Therefore, we developed a system for video streaming from sketch, using open-source X.264 implementation of H.264 and UDP/IP stack. We modified the X.264 codec to include adaptive controls at the decoder side to come up with an effective trade-off between low latency (achieved by reducing buffer occupancy) and video fluency (achieved by avoiding buffer occupancy to go to zero). Our extensive experimentation proves the effectiveness of our solution.

As future directions of our research, we will implement the X.264 decoder on a mobile device, such as a PDA, to also test the feasibility of our solution on devices with low computational power. Moreover, more and long-lasting experiments will be carried out to evaluate the reliability of the system. Eventually, the implementation also of the X.264



(a) @ 23 kbps

(b) @ 80 kbps

(c) @ 120 kbps

Figure 5. Achieved video quality (measured with PSNR) for our system with a bandwidth of 23 kbps (a), 80 kbps (b) and 120 kbps (c). Average PSNR values are 31.00 dB (@ 23 kbps), 33.94 dB (@ 80 kbps) and 34.94 dB (@ 120 kbps).

encoder on a PDA will be considered, even though this is a quite challenging direction.

Acknowledgments

This work is partially supported by the project L.A.I.C.A. (Laboratorio di Ambient Intelligence per una Città Amica), funded by the Regione Emilia-Romagna, Italy, and European VI FP, Network of Excellence DELOS (2004-08) on digital libraries, sub-project Multimedia Interfaces for Mobile Applications (MIMA).

References

[1] Advanced video coding for generic audiovisual services. Technical report, ITU Rec. H624/ISO IEC 14996-10 AVC, 2003.

[2] http://developer.apple.com/opensource/server/streaming/index.html, Last accessed: 11 August 2006.

[3] http://developers.videolan.org/x264.html, Last accessed: 11 August 2006.

[4] http://sourceforge.net/projects/ffmpeg/, Last accessed: 11 August 2006.

[5] http://www.microsoft.com/windows/windowsmedia/, Last accessed: 11 August 2006.

[6] http://www.realnetworks.com/products/media_delivery.html, Last accessed: 11 August 2006.

[7] http://www.videolan.org/, Last accessed: 11 August 2006.

[8] A. Argyriou. A novel end-to-end architecture for H.264 video streaming over the internet. *Telecommunication Systems*, 28(2):133–150, 2005.

[9] A. Argyriou and V. Madisetti. Streaming h.264/avc video over the internet. In *Proc. of 1st IEEE Consumer Communications and Networking Conference*, pages 169–174, 2004.

[10] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik. Video coding for streaming media delivery on the internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):269–281, Mar. 2001.

[11] M. Guo, M. Ammar, and E. Zegura. V3: a vehicle-to-vehicle live video streaming architecture. In *Proc. of IEEE Intl Conf on Pervasive Computing and Communications*, pages 171–180, 2005.

[12] K. Lim, D. Wu, S. Wu, R. Susanto, X. Lin, L. Jiang, R. Yu, F. Pan, Z. Li, S. Yao, G. Feng, and C. Ko. Video streaming on embedded devices through GPRS network. In *Proc. of IEEE Intl Conference on Multimedia and Expo*, volume 2, pages 169–172, 2003.

[13] Z. Liu and G. He. An embedded adaptive live video transmission system over GPRS/CDMA network. In *Proc. of Intl Conf. on Embedded Software and Systems*, 2005.

[14] J. Lu. Signal processing for internet video streaming - a review. In *Proc. of Conf on Image and video communications and processing*, pages 246–259, 2000.

[15] M.-T. Lu, C.-K. Lin, J. Yao, and H. Chen. Complexity-aware live streaming system. In *Proc. of IEEE Int'l Conference on Image Processing*, volume 1, pages 193–196, 2005.

[16] A. MacAulay, B. Felts, and Y. Fisher. WHITEPAPER IP streaming of MPEG-4: Native RTP vs MPEG-2 transport stream. Technical report, Envivio, Inc., Oct. 2005.

[17] A. Puri, X. Chen, and A. Luthra. Video coding using the H.264/MPEG-4 AVC compression standard. *Signal Processing: Image Communication*, 19:793–849, 2004.

[18] T. Stockhammer and M. Hannuksela. H.264/AVC video for wireless transmission. *IEEE Wireless Communications*, 12(4):6–13, Aug. 2005.

[19] T. Stockhammer, M. Hannuksela, and T. Wiegand. H.264/AVC in wireless environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):657–673, 2003.

[20] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), July 2003.