

Concepts for Exchanging Extra Information Between Protocol Layers Transparently for the Standard Protocol Stack

Sandrine Mériegeault* - Catherine Lamy**
Philips Recherche France (PRF)
51 rue Carnot, B.P. 301
92156 Suresnes Cedex, France
{sandrine.merigeault, catherine.lamy}@philips.com

Abstract—Popular applications such as conferencing or audio/video streaming over wireless networks require the transmission of data with real-time and bandwidth constraints. Due to these strong constraints, optimizing the end-to-end transmission is almost a necessity. However, applications do not deal directly with the physical level but through protocol networks, and the available transmission methods depend only on these protocols, without any respect to the considered application. In order to allow optimisation of application level source coding and FEC at the network access layer in a standard protocol stack, two adaptation layers are inserted, namely source adaptation layer and channel adaptation layer, that are used to sort of bypass the internet and transport layers.

Index Terms—Network transparency, information exchange, adaptation layers, real-time transmission, joint source channel (de)coding.

I. INTRODUCTION

A protocol stack in a network is organized as a set of layers, each one built upon its predecessor as defined by the OSI (Open System Interconnection) reference model. The function of each layer differs from network to network, but in every network, the purpose of each layer is to offer given services to its neighbour layers. As a consequence, protocol layers that are not juxtaposed cannot exchange information. Such a restriction may reveal itself limitative.

For instance, high data compression and robust transmission often meet conflicting requirements, in particular when a bandwidth limited wireless link is considered. Indeed, transmission systems operating over bandwidth-limited channels require source encoding to compress the bitstream by reducing the source symbols inherent redundancy. Unfortunately, compression techniques that provide highly compressed sources allowing for bandwidth-efficient transmissions also generate bitstreams very sensitive to channel perturbations. Channel coding, in the form of Forward Error Correction (FEC), is thus generally applied to provide an appropriate level of protection, at the price of bandwidth expansion. This explains why a trade-off must generally be found between the amount of bandwidth allocated to the information data, and the amount that is allocated to the error protection techniques. In practice however, such trade-offs are only roughly determined, as applications do not directly deal with the network access layer, which is the combination of the physical and data link layers in the TCP/IP protocol model, but through network protocols. Indeed, actual source and channel en/decoders are often net-

worked devices that cannot directly exchange information due to the standardized protocol layers that separate them.

Hence, the available protection methods depend mainly on the considered protocol, without any respect to the considered application. For transmissions using Internet protocols, many solutions have been proposed and are currently deployed or under study. Typical protocols are based on the IETF protocol TCP, UDP, and the real-time transmission dedicated Real-time Transfer Protocol (RTP). A better trade-off can however be obtained by using Joint Source Channel Coding (JSCC) techniques [1], where both encoders and decoders aim at working together for improving their performance. In fact, such an approach not only allows to adapt the transmission to the considered application, but also permits to finely attune the transmission parameters (encoding rates, chosen FEC) to the current transmission conditions.

As stated above, such an efficient communication between two separate protocol layers is unfortunately not possible if one wants to avoid any redefinition of the existing protocols, first to keep the system backward compatibility and second to allow the implementation of this improvement on as many existing systems as possible. As a consequence, two solutions appear possible: either bypassing the network protocol stacks, or "fooling" it with transmitting syntactically correct packets that contain the considered extra information. The first case corresponds to the transmission of the supplementary information via a link parallel to the protocol stack. As a matter of fact, such a transfer cannot be interpreted by the various protocol layers, hence it can only be done locally, for instance by means of a dedicated driver [2].

The aim of the work presented in this paper corresponds to the second case. A concept for allowing the joint optimisation of application and network access layers transparently for the standard protocol stack is proposed, in the context of real-time data transmission over a mobile network.

This article is organized as follows. Section II presents the transmission scheme and describes the information exchanges than can take place between the network access and application layers. Section III presents the concept to realize said information exchanges, introducing two adaptation layers to this effect. The feasibility of the concept has been studied in the context of a RTP/UDP/IP protocol stack. Finally, Section IV draws out the conclusions.

II. TRANSMISSION SCHEME

Among the possible useful information exchanges between the source and channel coders, *i.e.*, between the appli-

* S. Mériegeault is now with Philips Digital System Labs - Paris, 51 rue Carnot, B.P. 301 92156 Suresnes Cedex, France.

** C. Lamy is now with THALES Communications France, 66 rue du Fossé Blanc, B.P. 156 92231 Gennevilliers cedex, France (e-mail: catherine.lamy@fr.thalesgroup.com).

cation and network access layers, we first find at the transmitter side the *Source Significance Information* (SSI). This information on the considered source, for instance the sensitiveness of the source symbols to channel errors, will allow to use efficient *Unequal Error Protection* (UEP) [3][4]. At the receiver side, the *Channel State Information* (CSI) can also be useful to adapt the source and channel coding rates to channel conditions [5]. A possible use of such CSI at the application level is to implement specific retransmission strategies depending on the quality of the channel link (request retransmission, apply concealment, . . .). And finally, in order to attune the source and channel decoding at the receiver side, it is interesting to provide information about the pertinence of the channel decoding to the source decoder. This channel decoder soft output, or *Decoder Reliability Information* (DRI), will allow to perform soft-output VLC decoding [6], [7].

The bandwidth needed to transmit extra information varies with the type of this information: whereas it may consist of only a few bits for the SSI and the CSI, the DRI will easily multiply the necessary bandwidth by four or five. Moreover, it is well known that wired transmissions do suffer much less from bandwidth limitations than wireless ones, and are definitely less prone to errors. As a consequence, the aforementioned information exchanges are especially interesting in the wireless case, and we will consider such a case in this paper.

Fig. 1 and Fig. 2 illustrate the possible extra information exchanges between the application and the network access layers during a transmission between a mobile and a server in the context of real-time data transmission over a mobile network, for both data upload or data download.

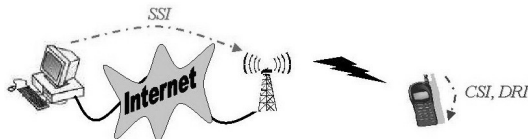


Fig. 1 Transmission scheme for data upload.

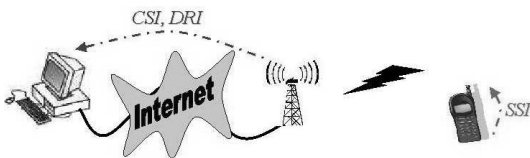


Fig. 2 Transmission scheme for data download.

Let us consider more closely the data download case as an example. The SSI is transmitted from the source encoder within the mobile to its channel encoder. As both decoders are in the same equipment, the link is local and a bypass can be created via a driver. On the other hand, the CSI and the DRI obtained after the wireless link at the channel decoder level can be used in the source decoder. However, those two decoders are separated by the Internet and several protocol layers, which implies the need to implement an adequate routing process to dispatch the extra information towards its goal. It will consequently be sometimes possible to transmit

locally the extra information, whereas at other times this information will need to be transmitted through an entire network, as illustrated in Fig. 3 in the case of data download. Refer to Appendix for the detailed traditional protocols used in a real-time data transmission stack.

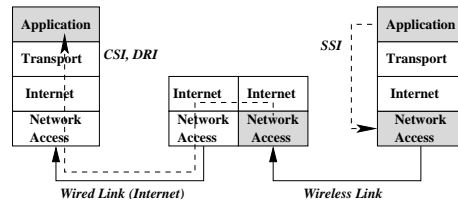


Fig. 3 Network scheme for data download.

III. ADAPTATION LAYERS CONCEPT

The goal of this paper is to propose a solution that allows the exchanges *between the application and the network access layers* through the protocol stack *in a transparent way*, i.e. without interfering with the regular use of the network, in order to keep the Quality of Service (QoS) of the original packet. As a consequence, we propose to introduce both at the receiver and transmitter sides adaptation layers. These layers will permit to successfully transmit information through the standard network protocol stack by “fooling” it. As shown in Fig. 4, these adaptation layers consist on the first hand of a **Source Adaptation Layer (SAL)**, which will be the interface between the application layer and the rest of the protocol stack and on the other hand of a **Channel Adaptation Layer (CAL)**, which will be the interface between the network access layer and the rest of the protocol stack.

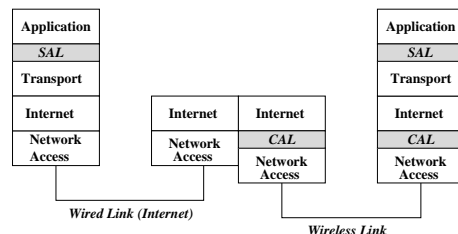


Fig. 4 System for the Network Transparency concept.

The adaptation layers are meant to work together, with the first one generating the extra packet(s) containing the supplementary information, which are later interpreted and re-organized by the second one. This naturally leads to the necessity of defining a marking means at the generating layer for the supplementary information to be reconnected to the original packet at the interpreting layer. Generating extra packets that are correct from a protocol stack point of view presents the main advantage to allow the transmission of the said packets over an entire network system.

Let us now consider how the extra packets can be used while keeping the Quality of Service offered by the protocol stack. In real-time data transmissions, the QoS is provided among others by the Real-time Transport Control Protocol (RTCP) which periodically sends feedback control messages to the transmitter side about the received data. Consequently, any supplementary information sent by the CAL to

the SAL through the protocol stack will lead to corresponding additional RTCP messages automatically generated at the transport layer and sent back to the transmitter. Such additional messages would disturb the emitting host, as it would not understand messages corresponding to supplementary information it never sent. To avoid this, a first solution would consist in deleting all the feedback messages, leading to a complete loss of the QoS. A second solution consists in transmitting the extra packets via a distinct port number in the transport protocol header. In this case, the original port number will correspond to the transport of the original packet, where the protocol stack QoS mechanisms will be active, *i.e.* where the RTCP feedback control messages will be sent back to the transmitter side. A second port number will be used to transport the supplementary information. Known by the CAL which will update accordingly the transport protocol header field of the extra packets (see Section III-A), this RTP port number will correspond to an RTCP one (the next higher (odd) port number [8]) where any RTCP packet is deleted at the network access layer. This way, the CAL can provide extra information to the SAL while the standard QoS mechanisms are kept active.

A. Transmission From the CAL to the SAL

Let consider the transmission of extra information from the CAL, placed at the transmitting or receiving side, to the SAL at receiving side. As illustrated by Fig. 5, this consists of four main steps, namely: the standard transmission of the original packet, the generation of the extra packet containing the supplementary information and an adequate marker, the transmission of the extra packet and finally the exploitation of the supplementary information.

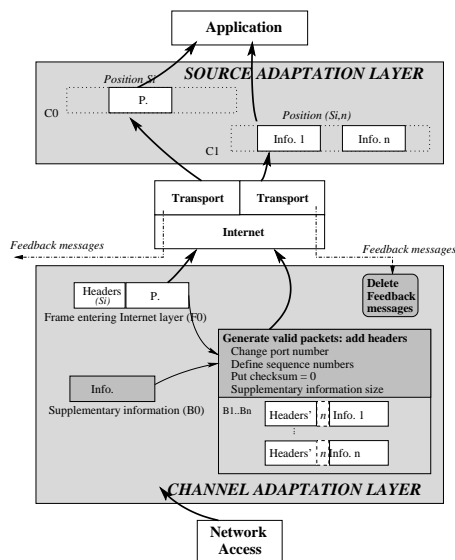


Fig. 5 Transmission of extra information from the CAL to the SAL.

During the first step, corresponding to the *standard transmission*, the CAL extracts from the data provided by the radio link (*e.g.* channel decoder) the original packet constituted of headers and payload as classically done and place it into a buffer **F0**. Note that this original packet may have been corrupted by the transmission, hence the data it contains may be erroneous. If the packet headers are correct, the

packet is transmitted through the whole protocol stack and is received at the application layer by the SAL. Otherwise the packet is discarded and, depending on the considered network protocols, a retransmission may occur later. This may be a problem for some applications, such as soft-input source decoding. In this case, it can be proposed to modify the headers of the original packet by setting the UDP checksum to zero, to ensure that even if their payload is corrupted, packets may still reach the SAL. The QoS of the protocol stack is preserved by keeping the feedback control messages. The payload is stored in the SAL buffer **C0** at the position given by the sequence number in the transport protocol header.

In the second step, denoted *extra packet generation*, the CAL extracts from the network access layer the supplementary information necessary for the application layer. Depending of the supplementary information size, one or several extra packets are created at the CAL level in the buffers **B1..Bn**. Each of these packets contains the supplementary information **B0** wholly or partly and the number of extra packets if there are more than one. Valid headers are then generated, taking into account fields such as the setting of the specific transmission port number and elements such as the size of the extra information. The sequence number of the extra packet is derived from the one of the original packet, to allow the SAL to be able to link them at the application level. In the case where several extra packets are created, the sequence number will be a combination of the original packet sequence number with the number of extra packets. Note that it is necessary to reset or re-compute the checksum fields in order to force the transmission of this supplementary information. Note that if the original packet is fragmented, it can be judicious to create extra packet(s) only for the first fragment or for each fragment by taking into account the fragment number or by using in the headers of the extra packets the same fragmentation fields as in the original packet. Finally, the CAL is set to delete all the feedback messages corresponding to the transmission of the supplementary information.

In the third step, named *extra packet transmission*, the extra packets stored in the buffers **B1..Bn** are transmitted through the protocol stack and stored in the SAL buffer **C1** at a position given by their sequence number.

Finally, in the fourth and last step, called *supplementary information exploitation*, the SAL, having received the initial payload and the supplementary information, processes the extra information with its corresponding payload.

B. Transmission From the SAL to the CAL

Mirroring the transmission presented in Section III-A, we describe here the transmission of supplementary information from the SAL, either at the transmitting or receiving side to the CAL at receiving side. As illustrated by Fig. 6, this process consists also of four main steps.

In the first step, corresponding to the *standard transmission*, the SAL extracts from the application layer the payload and sends it through the protocol stack.

In the second step, denoted *extra packet generation*, the SAL receives from the application layer the extra informa-

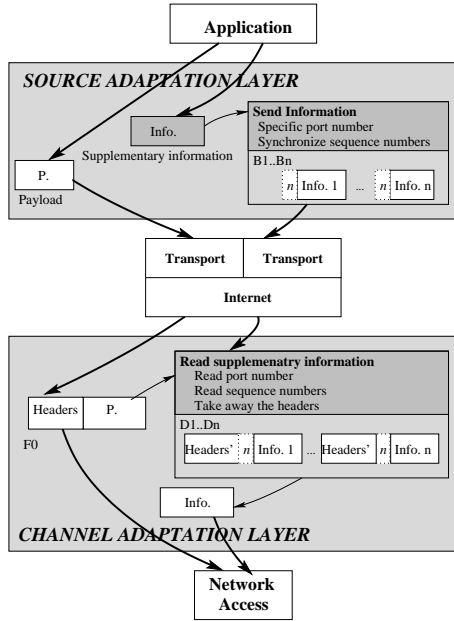


Fig. 6 Transmission of extra information from the SAL to the CAL.

tion necessary at the network access layer and creates accordingly one or several extra packets in the buffers $B1..Bn$, depending on the supplementary information size. Beside the extra information, these packets contain the number of extra packets (if there are more than one) and a marker, for instance the sequence number, that will allow the synchronization of initial payload and extra information at the CAL level. If several packets were created, the marker will take into account both the original packet sequence number and the number of extra packets. Note also that if the original packet is fragmented during the transmission, all the fragments and the supplementary information are still linked via the marker, but may require the re-assembly of the original packet to allow the use of the supplementary information over all the packet.

In the third step, called *extra packet transmission*, the extra packets stored in the buffers $B1..Bn$ are transmitted through a protocol stack with a specific port number. The extra packets are stored in the CAL buffers $D1..Dn$.

In the fourth and last step, named *supplementary information exploitation*, the CAL receives into a buffer $F0$ the original packet constituted by headers and payload, as classically done. Having received the extra packet(s), the CAL reads their headers in order to know if they are linked with the original packet. If it is the case, the headers of the extra packet(s) are deleted in order to transmit to the network access level only the supplementary information.

C. Discussion on the concept feasibility

To show the feasibility of the network transparency concept, one requires an access to real transmitted data both below the IP layer and above the RTP layer. It was consequently chosen to build a simulator in the BSD (Berkeley Software Distribution) Linux environment and to directly use the UDP/IP protocols of the Linux computer.

Having in principle only access to the user-level of the computer, but needing to access below the IP layer to im-

plement the *Channel Adaptation Layer* functionalities, we have created links between the Linux kernel and the user-level. To achieve this, the new protocol family introduced in the Linux kernel (post-2.2 releases) was used in combination with L_{PF} filters based on Berkeley Packet Filter [9]. Named PF_PACKET [10], this new protocol allows an application to send and to access packets directly at the network card driver level, thus avoiding the usual protocol stack and allowing to modify the various protocol fields by directly interacting with the Ethernet interface.

The resulting feasibility scheme is presented in Fig. 7. Covering only the transmission from the CAL to the SAL at the receiver side, it consists of one sender, two receivers and one *Channel Emulator Block*. The sender sends a data sequence by the means of datagram socket (SOCK_DGRAM), which relies on the UDP protocol services. This packet goes down the protocol stack to the Ethernet layer. The *Channel Emulator Block* has opened a PF_PACKET family socket to listen to traffic at the Ethernet level. A L_{PF} filter is attached to this socket and will transmit the previous packet encapsulated with the headers of the RTP/UDP/IP/Ethernet stack to the user-level. Noise is added to this packet by a channel simulator. Supplementary information can be extracted from the Ethernet level. The noisy packet and the supplementary information are transmitted to the *Channel Adaptation Layer*. At the output of this layer, the original packet and the extra packet are sent by a raw socket (SOCK_RAW) with the IP_HDRINCL socket option directly to the IP level. The payloads are then received respectively by the different receivers (one for each UDP port number).

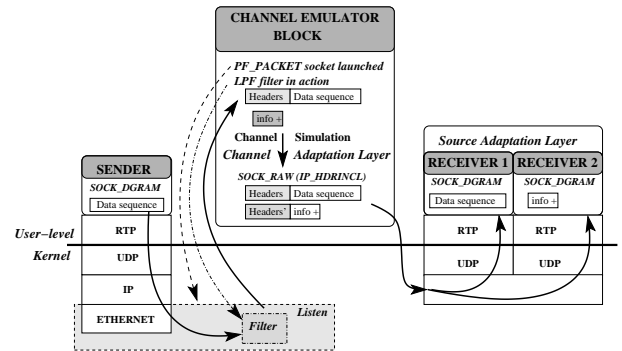


Fig. 7 Feasibility scheme of the network transparency concept.

IV. CONCLUSION

This article presents a concept for exchanging information between the application and network access layers in a transparent way through a network protocol stack for a real-time data wireless transmission. The solution consists in adding two adaptation layers to the network protocol stack: a *Channel Adaptation Layer* and a *Source Adaptation Layer* that generate and interpret extra packet(s) transporting the supplementary information.

An original technique has been developed that consists in generating valid packets containing the supplementary information in order to be compliant with the protocol stack and so to cross it. In order not to disturb the transmission of the original packet (*i.e.* to keep the QoS), a specific port

number is defined in the transport protocol header. This technique allows to transmit the supplementary information across an entire network (*i.e.* not only across one protocol stack) and can be used in combination with local bypass via means of drivers.

The feasibility of this concept has been studied at the receiver side by a simulation in a BSD Linux environment by implementing the *Channel Adaptation Layer*.

APPENDIX

In the case of real-time data transmission, considering Internet Protocol (IP) as network layer protocol, the most prevalent protocol is the Real-time Transport Protocol (RTP) [8], which is usually used in combination with the User Datagram Protocol (UDP). In such a case, the Quality of Service (QoS) is provided among others by the Real-time Transport Control Protocol (RTCP) [8]. RTCP periodically sends feedback control messages to the transmitter side about the received data. The explicit standardized headers for these three protocols [11] are detailed in Fig. 8.

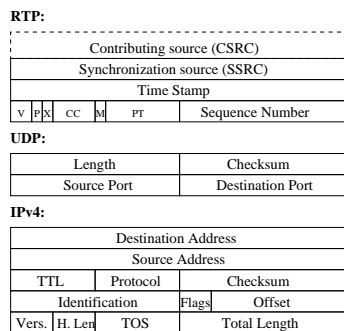


Fig. 8 Headers for the RTP/UDP/IPv4 stack.

REFERENCES

- [1] L. Camiciotti, C. Lamy, L. Meilhac, S. Olivieri, and P. Verdi. "Joint source-channel coding for 4G multimedia streaming". In *2nd WWRF meeting, WG3*, Helsinki, Finland, May 2001. <http://www.wireless-world-research.org/WWRF2/wwrf2.html>.
- [2] A. Rubini. *LINUX Device Drivers*. O'Reilly and Associates, 1998.
- [3] K. Fazel and J.J. Lhuillier. "Application of unequal error protection codes on combined source-channel coding of images". In *Proc. ICC'90*, volume 3, pages 898–903, Atlanta, USA, April 1990.
- [4] M. Gallant and F. Kossentini. "Rate-distortion optimized layered coding with unequal error protection for robust Internet video". *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):357–372, March 2001.
- [5] I. Kozintsev and K. Ramchandran. "Robust image transmission over energy-constrained time-varying channels using multiresolution joint source-channel coding". *IEEE Trans. Signal Processing*, 46(4):1012–1026, April 1998.
- [6] M. Bystrom, S. Kaiser, and A. Kopansky. "Soft source decoding with applications". *IEEE Trans. Circuits and Systems for Video Technology*, 11(10):1108–1120, Oct. 2001.
- [7] L. Perros-Meilhac and C. Lamy. "Huffman tree based metric derivation for a low-complexity sequential soft VLC decoding". In *Proc. ICC'02*, volume 2, pages 783–787, New York, USA, Apr.-May 2002.
- [8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RFC1889: RTP: A Transport Protocol for Real-Time Applications". <http://www.ietf.org/rfc/rfc1889.txt>, Jan. 1996.
- [9] S. McCanne and V. Jacobson. "The BSD Packet Filter: A new Architecture for User level Packet Capture". In *Proc. Winter'93 USENIX*, pages 259–269, San Diego, USA, Jan. 1993.
- [10] G. Insolvibile. "Kernel Korner: The Linux Socket Filter: Sniffing Bytes over the Network". *Linux Journal*, 86, June 2001.
- [11] W. Richard Stevens. *TCP/IP Illustrated Volume 1: The Protocols*. Reading: Addison-Wesley Corporate & Professional, 1999.