

# A Versatile Multiplexing Algorithm Exclusively Based on the MPEG-2 Systems Layer

Samuel Rivas

LambdaStream Servicios Interactivos S.L.  
Ronda de Outeiro 33 Entlo., A Coruña (Spain)  
Email: samuel@lambdastream.com

Víctor M. Gulías and Carlos Abalde

LFCIA - MADS Group. Computer Science Department  
University of A Coruña (Spain)  
Email: {gulias,carlos}@dc.fi.udc.es

**Abstract**—This work introduces an algorithm to multiplex stored MPEG-2 PES streams into one SPTS regardless of the PES streams type or bitrate. The algorithm is devised to comply with the MPEG2 standard STD. In order to support all available levels and profiles, the STD buffer lengths are parametrized and a maximum output bitrate can be established. To check the validity of the algorithm, a simulator has been implemented in a high level functional language. To prove its usefulness, a MPEG-2 PS to TS converter was implemented in C using this algorithm. The paper focuses on the algorithm description and shows some results obtained with the simulator.

## I. INTRODUCTION

In a video server [1], where different types of multimedia data are stored in PESs (Packetized Elementary Streams), or even grouped in single programs in PSs (Program Streams), it is necessary to combine a set of them into one SPTS (Single Program Transport Stream) to serve it when a user sends a request to the server. This SPTS must be compliant with the STD (System Target Decoder) defined in the MPEG-2 systems standard [2] in order to avoid buffer overflows or underflows in later stream operations (such as demultiplexing and decoding it).

Many studies address the problem of remultiplexing several input TSs (Transport Streams) into one single MPTS for applications such as digital TV. Those works face three main problems: data insertion, PID (Packet Identifier) mapping and PCR (Program Clock References) correction [3]–[6]. The main goal of these works is to multiplex a number of inputs so that they fit in a given output bitrate, possibly transcoding the input streams to adapt their bitrates [7]. Another common feature is that they receive the input data in real time in TS form or directly from the MPEG encoder [8], so they can implement a scheduling algorithm [9] to multiplex the packets they have in their input buffers.

When the media is stored and is not in TS format, the problem needs to be addressed in a different way:

- As long as we do not have PCR information, the problem of PCR correction turns into PCR generation.
- The only temporal information about input packets is provided by the PTS (Presentation Time Stamp) and DTS (Decoding Time Stamp) and the scheduling algorithm cannot be based on arrival times.
- The output bitrate cannot be based on the input bitrates, since input bitrate can be much higher (reading from

a disk is faster than reading from a real time stream). Since we want to be able to remultiplex any PES stream regardless of its content, output bitrate must be calculated upon input stream timestamps at the MPEG-2 systems layer.

This paper is organized as follows. In section II an introduction to the main problems to be solved by the proposed algorithm is presented. Next, section III describes the different solutions adopted to tackle each of the enumerated problems and how those solutions are glued to build the whole algorithm. Then, section IV shows an overview of some results over a simulator of the algorithm developed in a high level functional language (Erlang [10]) to fast check the algorithm, collecting statistics and graphs. Finally section V shows some conclusions and future research directions.

## II. PROBLEM OVERVIEW

### A. Time Simulation

The algorithm should return a TS which never overflows nor underflows the STD defined in the MPEG-2 standard [2]. To achieve this, we record the status of an hypothetic STD when multiplexing the inputs. Hence, the PCR values are generated and, at the same time, we simulate the behaviour of a STD to ensure the correctness of the generated stream.

Our algorithm works with a given PCR period  $T$  and it simulates the pass of time within each interval  $[PCR_t, PCR_t + T)$  according with a calculated bitrate for it. Henceforth, we will refer to a interval like that as *the step t*. Since we are simulating the encoder time and assuming a constant bitrate for each algorithm step, we can know what the encoder clock value should be for each output TS packet. Along this paper we would refer to this value as VPCR (*Virtual PCR*).

### B. Data Insertion

Having the VPCR value, PSI (Programme Specific Information) insertion is a straightforward task. The algorithm has two parameters to specify the PAT (Programme Association Table) interval ( $PAT\_T$ ) and the PMT (Programme Map Table) interval ( $PMT\_T$ ). A PAT or a PMT must be written when the corresponding condition becomes true:

$$VPCR - VPCR_{PAT} > PAT\_T \quad (1)$$

$$VPCR - VPCR_{PMT} > PMT\_T \quad (2)$$

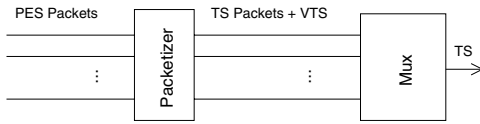


Fig. 1. Main algorithm modules

Where  $VPCR_{PAT}$  is the VPCR value when last PAT was inserted and  $VPCR_{PMT}$  is the VPCR value when last PMT was inserted.

### C. PID Mapping

Since we are going to create a SPTS, PID mapping is a very easy task. Each input stream will be assigned to a different PID. All these PIDs are listed in the PMT grouped in one single program.

### D. Packetization

Before multiplexing, PES packets must be cut up and packetized in TS packets. A TS packet has a length of 188 bytes, 4 of them are reserved for the header and the rest may be used to carry payload. Since a PES packet can be much longer than 184 bytes, the packetizer should be able to transform each PES into a sequence of TS packets.

We separate the packetizing step from the muxer algorithm itself. The packetizer must assign a *Virtual Time Stamp* (henceforth VTS) to each TS packet, so that the muxer can estimate the time when that packet is going to be removed from the STD buffer. The muxer algorithm works with N inputs of TS packets where each TS packet is labeled with a VTS as shown in fig. 1.

### E. STD Meeting

Each input stream will have an associated STD buffer size. MPEG-2 standard asserts certain values for video and audio streams depending on profile and level. We generalise our algorithm setting a variable called  $B_i$  to denote the buffer size for each input stream  $i$ .

When the muxer algorithm outputs a TS packet with data of an input stream, it should be sure that this packet will not overflow the STD buffer. For this purpose, the algorithm records information about emitted packets for all input streams. This information is modeled as a list of pairs  $L_i$ <sup>1</sup>:

$$L_i = [\{b_1, VTS_1\}, \{b_2, VTS_2\}, \dots]$$

Where  $b_j$  is the payload length of the TS packet and  $VTS_j$  is the VTS for that packet.

At any virtual time  $VPCR$ , the system must comply with the following condition for all input stream  $i$ :

$$B_i \geq \sum b_j | \{b_j, VTS_j\} \in L_i \wedge VTS_j \geq VPCR \quad (3)$$

When trying to meet that condition, a problem arises. Between two PCR values in the generated TS, the bitrate is

<sup>1</sup>Note that not every TS packet carries 184 bytes of payload. A TS packet with last data chunk of a PES packet may be filled with stuffing, which does not enter in the decoding buffer.

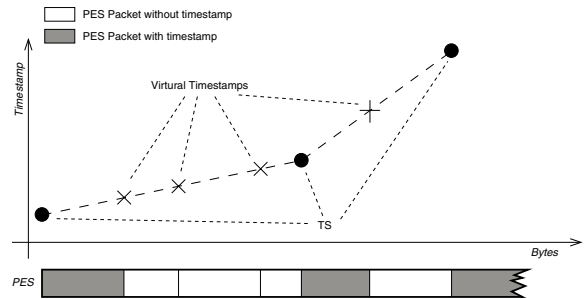


Fig. 2. VTS values interpolation

assumed to be constant. This fact is used to work out the VPCR value for each inserted TS packet. With the VPCR value, we can test if the insertion of a packet in the output TS stream will break condition 3 and, in that case we should avoid the insertion in favour of a packet with data of other input stream. However, to know the bitrate we need to know how many packets are going to be inserted and, here, a circular dependence appears. To solve this, the number of packets that will be inserted in each PCR to PCR period must be somehow estimated.

## III. ALGORITHM DESCRIPTION

The algorithm works with any number of input streams; along this paper we will refer the amount of input streams as  $I$ . Next sections describe the strategies used to solve the arisen problems.

### A. Packetization Process

The main packetizer task is to cut up the input PES packets and to fit them into a sequence of TS packets. This is a task that requires only to take care of the continuity counter values, the payload unit start indicator and the needed stuffing for the last TS packet of a given PES packet. In addition, the packetizer must assure that any output TS packet is associated with a VTS value, so that the multiplexing algorithm could work.

A PES packet can carry two different timestamps: PTS and DTS. For PES packets with timestamp information, that value is used as VTS, choosing the DTS if present or the PTS in other case<sup>2</sup>.

Let  $TS_0$  be the associated timestamp of a given PES packet  $PES_0$  and let  $PES_n$  be the first PES packet with timestamp information just after  $PES_0$ . For each PES packet  $PES_j$  between them, the VTS value is calculated as follows:

$$\forall j \in (0, n), VTS_j = TS_0 + \frac{TS_n - TS_0}{\sum_{k=0}^{n-1} l_k} \sum_{y=0}^{j-1} l_y$$

Where  $l_y$  is the total length of  $PES_y$  and  $TS_n$  is the PTS or DTS in  $PES_n$ . Fig. 2 depicts a visual scheme of this formula.

<sup>2</sup>Because of this, all streams must convey the PES packets in decoding order.

## B. Virtual Time Management

One algorithm step comprises from the writing of a PCR packet to the writing of the TS packet just before next PCR packet. Along one step, the bitrate used to calculate the VPCR is assumed to be constant, so the operation to keep VPCR up to date consist in adding  $\frac{188}{\text{bitrate}}$  to it after writing each packet<sup>3</sup>.

## C. Bitrate Estimation

1) *Goals*: When remultiplexing, two main problems must be avoided: buffer overflows (condition 3) and underflows. Buffer overflows are avoided by the algorithm by means of its simulated STD, as we will describe later, and buffer underflows are avoided because the algorithm tries to multiplex the biggest amount of input TS packets (from the packetizer) meeting the precalculated bitrate and the STD constrains. Hence, to avoid buffer underflows a conservative allocation of bitrate has to be made for each step, understanding conservative as “better more than less”.

2) *Output packets calculation*: As we explain in section II-E, it is necessary to estimate the number of TS packets to be written in each algorithm step. To handle this task and avoid STD buffer overflows, the algorithm should hold information about  $L_i$  for each multiplexed stream.

With the information in  $L_i$ , we can estimate the number of packets to be written in each step. Let  $BD_{i,t}$  be “the amount of data that would be stored in the STD buffer at the end of the step  $t$  for the stream  $i$  if no more data were inserted”:

$$BD_{i,t} = \sum b_j |\{b_j, VTS_j\} \in L_i \wedge VTS_j > PCR_t + T$$

At the step  $t$ , the algorithm will be able to insert up to  $BD_{i,t}$  payload bytes for the stream  $i$ . With this information and inspecting the input TS packets buffer, we can know how many packets can be written in this step at most for each stream. Let  $P_{i,t}^{pld}$  be the maximum number of packets that can be written for the stream  $i$  along the step  $t$ .

$$P_{i,t}^{pld} = \max n \left| \sum_0^n ib_j \leq B_i - BD_{i,t} \right. \quad (4)$$

Where  $ib_0, ib_1, \dots, ib_n$  are the payload lengths for the TS packets in the input buffer for the stream  $i$ .

Thus, the algorithm can output at most  $\sum_i P_{i,t}^{pld}$  packets with payload in the step  $t$ ; more packets will certainly overflow the STD buffer.

One additional packet must be written for the PCR time reference and up to two more packets may be needed if PSI data must be inserted along this step. Let  $P_t^{PSI}$  be the number of packets with PSI data required for step  $t$ . Then, the amount of packets to be output for the algorithm at step  $t$  can be estimated as follows:

$$P_t = 1 + P_t^{PSI} + \sum_{i=1}^I P_{i,t}^{pld}$$

<sup>3</sup>MPEG-2 handles the time in tics of 90 Khz (ignoring PCR extension), so we manage the bitrate in units of bytes per tic.

Assuming that  $PAT\_T \geq PCR\_T$  and  $PMT\_T \geq PCR\_T$  the  $P_t^{PSI}$  value is either two if both conditions (1) and (2) will be true at  $VPCR_t + PCR\_T$ , one if only one of them will be true or zero otherwise.

3) *Avoiding NULL packets*: As  $P_t$  is an upper limit, choosing it as the amount of packets to be output for the step  $t$  ensures that the algorithm does the best it can to avoid buffer underflows. But this value has an inconvenience: it represents the amount of packets that can be written *at the end of the step*, but it is not assured that a TS packet can be output at any time along the step. When the algorithm reaches a point where it cannot output any TS packet, it must output a NULL packet.

At the simulation stage, we appreciated that using the bare  $P_t$  value will cause an unbearable amount of NULL packets at the output stream for most of the cases, so a little correction must be done.

Instead of using all the available STD buffer to work out the number of packets that can be output along an algorithm step, a smaller value is used. As a result, the algorithm will try to keep the STD buffer at certain level. Let  $B'_i$  be that value for each stream  $i$ . The algorithm will estimate the bitrate for each step in order to comply with the following expression instead of with (3):

$$B'_i \geq \sum b_j |\{b_j, VTS_j\} \in L_i \wedge VTS_j \geq vt \quad (5)$$

However, along the step  $t$  it will be able to use the extra free space  $B_i - B'_i$  to insert those packets that would not be possible to insert if we used the  $P_t$  value.

Let be  $P_i^{pld}$  the same as  $P_i^{pld}$  but using  $B'_i$  instead of  $B_i$ . The algorithm will use a value  $P'_t$  to work out the bitrate for the step  $t$ .

$$P'_t = 1 + P_t^{PSI} + \sum_{i=1}^I P_i^{pld}$$

The value  $B'_i$  must be less or equal to  $B_i$  so that  $P'_t \leq P_t$ .

4) *Bitrate limitation*: Another issue to be taken into account is the upper limit of the bitrate value. We define a parameter  $br^{max}$  to avoid unbearable peak values in the bitrate. This value is specially useful in the first steps of the algorithm when the STD buffers are empty. At this stage, with the method explained above, the algorithm will output all the packets needed to fill the buffers up to their limit  $B'_i$  at one only step. Let  $br_t$  be the assumed bitrate for the step  $t$ , using  $P'_t$  as the number of packets to output along the step  $t$ , the bitrate for the first algorithm step can be approximated as the next value, which is not related to the input bitrates but to the number and type of input streams.

$$br_t \approx \frac{\sum_{i=0}^I B'_i}{T}$$

According to the upper bitrate limit, the maximum number of packets that the algorithm can output in a single step is calculated as follows:

$$P^{max} = \left\lfloor \frac{br^{max} \times T}{188} \right\rfloor$$

Hence the number of output packets and the bitrate assumed for the step  $t$  are calculated as follows:

$$\begin{aligned} \bar{P}_t &= \min(P^{max}, P'_t) \\ br_t &= \frac{\bar{P}_t \times 188}{T} \end{aligned} \quad (6)$$

#### D. Output Packet Scheduling

Within each step the algorithm should choose the TS packet to output for each substep. As we have seen, one algorithm step outputs a precalculated number of packets and advances the virtual from a value  $PCR_t$  to  $PCR_t + PCR\_T$ , so we can think that we have an array of  $\bar{P}_t$  slots with a  $VPCR_{t,k}$  value assigned to each of them. For all  $k = 0, 2, \dots, \bar{P}_t - 1$ , VPCR is calculated as follows:

$$\begin{aligned} VPCR_{t,0} &= PCR_t \\ VPCR_{t,k} &= VPCR_{t,k-1} + \frac{188}{br_t} \end{aligned} \quad (7)$$

Next desired conditions hold:

$$\begin{aligned} \forall k \in [0, \bar{P}_t - 1] \quad VPCR_{t,k} &\in [PCR_t, PCR_t + T) \\ \forall k \in [0, \bar{P}_t - 2] \quad VPCR_{t,k+1} &> VPCR_{t,k} \end{aligned}$$

In addition, VPCR complies with the following condition which assures that VPCR continuously simulates the time suggested by the inserted PCR packets:

$$VPCR_{t, \bar{P}_t - 1} + \frac{188}{br_t} = PCR_{t+1}$$

With this information for any step  $t$  and instant  $k$ , the packet scheduler selects an output following the next scheme, considering that for each input stream  $i$  we have a pair  $\{b_i, VTS_i\}$  which represents the payload length and VTS for the next TS packet at the input buffer  $i$ .

- If  $VPCR_{t,k} - VPCR_{PAT} \geq PAT\_T$  then choose to output the PAT packet.
- Else if  $VPCR_{t,k} - VPCR_{PMT} \geq PMT\_T$  then choose to output the PMT packet.
- Else perform two steps, if needed, to choose the most suitable TS packet to output from the input streams:
  - 1) Construct a set with the values  $VTS_i$  for those streams so that inserting the pair  $\{b_i, VTS_i\}$  in  $L_i$  does not break the condition (5), being  $VPCR = VPCR_{t,k}$ , and return the stream with the smallest VTS value in this set.
  - 2) If the set constructed above were empty, construct a new one but relaxing to condition (3) and return the stream with the smallest VTS value in this new set.
- If both sets were empty then output a NULL packet.

Step 1 in the two-step subselection is not needed to output a STD compliant stream, but as proved with the simulator in section IV-B, it is needed to keep the STD buffer fullness near to  $B'_i$  for all input streams.

#### E. First PCR Value

When the algorithm begins, a suitable value for  $PCR_0$  must be chosen depending on the VTS assigned to the TS packets at the input buffers. We work out a PCR value so that the first packet that enters the STD buffer will be ready to be removed in the moment that all the input buffers are filled up to their  $B'_i$  value. Assuming this PCR origin, the algorithm generates an output stream at the maximum bitrate until the STD buffers reach the working level. From this moment, the algorithm operates at the bitrate dictated by the input streams.

To work out this value, we calculate how many steps are needed to fill all the STD buffers up to their associated  $B'_i$ . Let  $VTS_{min}$  be the lowest VTS in all the input buffers, if we calculate first PCR value as shown below, the first inserted TS packet will be ready to exit the STD buffer when all the buffers were approximately at its associated working level.

$$PCR_0 = \max \left( 0, VTS_{min} - \frac{\sum_{i=0}^I B'_i}{br_{max}} \right) \quad (8)$$

Note that in this equation we are not taking into account the PCR and PSI packets, nor that the header and stuffing of the TS packets with payload does not enter the STD buffers. Normally the input streams total bitrate is much higher than the bitrate needed for that kind of data so this approximation is good enough for a wide range of applications.

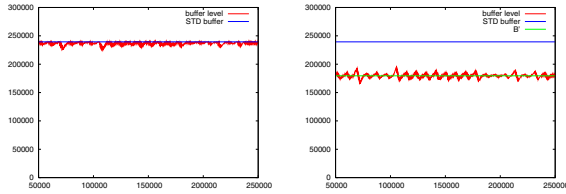
#### F. Overall Description

The overall algorithm skeleton is shown below:

- 1) Calculate first PCR value as explained in (8) and store it as  $PCR_0$ . Set  $VPCR_{PAT} = VPCR_{PMT} = PCR_0$ .
- 2) Set  $t = 0$ .
- 3) Set  $k = 0$ .
- 4) Work out  $\bar{P}_t$  and  $br_t$  as explained in (6).
- 5) Output a PCR packet with  $VPCR_{t,0}$  value.
- 6) Let *stream* be the output of the stream selection scheduling algorithm.
  - a) If *stream* is *PAT*, *PMT* or *NULL* then write the required packet. If a *PAT* or a *PMT* were written update  $VPCR_{PAT}$  or  $VPCR_{PMT}$  to  $VPCR_{t,k}$ .
  - b) If *stream* is  $i$  then write the TS packet from the input buffer for the stream  $i$ , add a pair  $\{b, VTS\}$  to  $L_i$ , where  $b$  is the payload length of that TS packet and  $VTS$  is its associated VTS.
- 7) If  $\bar{P}_t$  packets have been written for this step  $t$  then set  $t = t + 1$  and go to 3, else set  $k = k + 1$  and go to 6.

## IV. EXPERIMENTAL RESULTS

We used four pairs of video and audio streams with different bitrates and coded with different strategies to generate the inputs for the experiments shown in this section. We experimented multiplexing them in different groups and with different options to ensure that the results comply with the expected according with the previous sections of this paper. We set the  $PCR\_T$  to 4500 tics (0,05 seconds) for all examples.



(a) Using the whole STD buffer

(b) Using  $B'_i = \frac{3B_i}{4}$

Fig. 3. Impact of the  $B'_i$  buffer usage limitation

#### A. The Use of $B'_i$

Running the algorithm over our tests cases with  $B'_i = B_i$  demonstrated that the  $B'_i$  mechanism is needed to produce output streams that do not waste a lot of bitrate with NULL packets. In fig. 3(a), we show the video buffer fullness when decoding a test case with two streams (video and audio) multiplexed with  $B'_i = B_i$ . In fig. 3(b), it is depicted the video buffer fullness when the same test case is multiplexed with  $B'_i < B_i$ . We can see that with a lower level of buffer fullness, the algorithm can use the  $B_i - B'_i$  margin to output that packets that come in too early in the algorithm step in substitution of the NULL packet that must be output in the first case.

In these experiments, we found that setting  $B'_i = \frac{3B_i}{4}$  the algorithm outputs a NULL packet free stream for all test cases. The tests with  $B'_i = B_i$  output streams with NULL packet ratios from 4.22 % to 15.29 %.

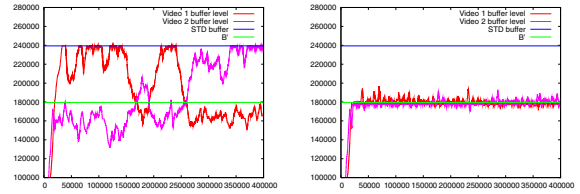
The NULL packet problem is mitigated, even without the use of  $B'_i$  when more than one video stream are multiplexed. The audio streams are not useful to solve the scheduling problems, since the audio buffer is too small and the audio bitrate is too low, the algorithm rarely can use an audio packet in substitution of a video packet that can not be output at certain moment.

#### B. The Two-step Scheduling Algorithm

In fig. 4, it is depicted how the two-step scheduling algorithm presented in III-D succeed in the task of keeping all multiplexed stream STD buffers at the  $B'_i$  level. If we used a one-step algorithm based only on the condition (3) one stream would be able to steal the packet slots reserved for the other until its maximum STD buffer level is reached.

#### C. PCR.T Value

We were using a fixed PCR.T value in all the previous experiments. The maximum allowed time between PCR references asserted in the MPEG standard is 0,1 seconds (i.e. 9000 tics). We checked that our algorithm worked well for values below this upper limit. Shorter values give more accuracy in the bitrate estimation but with higher computation overhead since a search in all input streams must be done for each step to work  $P_{i,t}^{pld}$  out. A shorter PCR.T value means also a wider wastage of bitrate in PCR packets since we are not using packets with payload for that purpose.



(a) One-step algorithm

(b) Two-step algorithm

Fig. 4. Behaviour of the two-step scheduling algorithm

## V. CONCLUSIONS

We developed an algorithm to multiplex multiple input streams regardless of their payload type or the way they are delivered to the multiplexor. The algorithm does not need any external clock reference rather than the input stream timestamps and it generates the PCR references based on a simulated STD. To check the algorithm behaviour, we implemented a simulator and applied it to several test cases. The results showed that the algorithm outputs a correct TS (in terms of clock references and timestamps) and the devised mechanism to control the decoder buffer levels worked as expected.

Finally, to check the usefulness of our algorithm, we used it as part of the implementation of a PS to TS converter implemented in C using our algorithm, where we had to face with the problem of buffering the input. Some additional work must be done here and it is proposed as a future research direction.

## REFERENCES

- [1] V. M. Gulías, M. Barreiro, and J. L. Freire, “VoDKA: Developing a video-on-demand server using distributed functional programming,” *Journal of Functional Programming*, vol. 15, no. 4, 2005.
- [2] *Generic Coding of Moving Pictures and Associated Audio Systems*, International Organisation for Standardisation ISO/IEC International Standard 13 818-1, Nov. 1994.
- [3] A. Ramaswamy and W. B. Mikhael, “Design of an efficient DVB/MPEG transport stream remultiplexor,” in *Proc. 43rd IEEE Midwest Symp. on Circuits and Systems*, vol. 2, Lansing MI, Aug. 8 – 11, 2000, pp. 754–757.
- [4] W. Xingdong, Y. Songyu, and L. Longfei, “Implementation of MPEG-2 transport stream remultiplexer for DTV broadcasting,” *IEEE Trans. Consumer Electron.*, vol. 48, no. 2, pp. 329–334, May 2002.
- [5] S. I. Lee, S. B. Cho, J. H. Kik, H. H. Jeon, and D. G. Oh, “Implementation of MPEG-2 TS remultiplexer and data transport unit for HDTV satellite broadcasting,” *IEEE Trans. Consumer Electron.*, vol. 43, no. 3, pp. 324–329, Aug. 1997.
- [6] L. Cheng, L. Chen, and X. Fang, “Design and implementation of transport stream remultiplexer with cascade architecture,” *IEEE Trans. Consumer Electron.*, vol. 49, no. 2, pp. 447–452, May 2003.
- [7] T. Takahashi, H. Kasai, T. Hanamura, M. Sugiura, and H. Tominaga, “MPEG-2 multi-program transport stream transcoder,” in *IEEE International Conference on Multimedia and Expo.*, Aug. 22 – 25, 2001, pp. 423–426.
- [8] S. J. Kim and J.-S. Koh, “An implementation of MPEG-2 transport stream multiplexer,” in *Proc. IEEE Workshop on Signal Processing Systems*, Oct. 20 – 22, 1999, pp. 379–389.
- [9] D. Jianghong, X. Zhongyang, C. Hao, and D. Hui, “Scheduling algorithm for MPEG-2 TS multiplexers in CATV networks,” *IEEE Trans. Broadcast.*, vol. 46, no. 4, pp. 294–295, Dec. 2000.
- [10] Open source erlang. [Online]. Available: <http://www.erlang.org/>