DVB-RCS2 Satellite Standard Performance Assessment Through a Link-Level Python Simulator

Rodrigo A. R. Fischer, João Paulo Leite

***Abstract*— The task of obtaining the performance of a communications receiver is difficult, since it involves the implementation of the physical layer transmission and reception chains and also requires great computational time, since Monte Carlo methods make up the base of link-level simulations. The DVB-RCS2 standard provides some receiver performance data which is rather restrict. This limits the capacity analysis made through a system level simulator. The objective of this work is to assess the DVB-RCS2 performance using a Python simulator developed by the authors for a much broader range of SNR values and for a comprehensive set of transmitter-receiver configurations.**

***Keywords*— DVB-RCS2, turbo coding, satellite communications, simulation.**

I. Introduction

Satellite systems have been broadly used by the communications industry. One example is the DVB-S system, that was developed to deliver digital television over satellite and serves over 100 million receivers [1]. The DVB-S defines only a broadcast channel, and was expanded (DVB-S2 and DVB-S2X) in order to be more robust and to adapt to the oncoming technologies. In response to requests from the industry, a return channel (DVB-RCS) was incorporated into the DVB standard to provide interactivity with the remote user. Further, the RCS standard evolved to DVB-RCS2, encompassing a mobility element, and more advanced network and physical layer techniques [1].

While deploying a new communications system technology, it is crucial to evaluate the performance of the proposed link interface. This information can be used not only to aid the development of the receiver terminal and characterize its capabilities, but also to feed higher level simulators, such as system simulators, with the physical link statistics. For the DVB-RCS2 system, the standard provides the user with the SNR required for only a chosen set of packet error rate (PER) performance, namely 10^{-3} and 10^{-5}, for the additive white gaussian noise (AWGN) channel [2].

This work presents the RCS2 receiver performance for a much broader range of SNR's, that can be used to allow a broader set of design rules through system level simulations. It also reviews key aspects of Turbo channel coding and decoding schemes and its implementation using the Python programming language.

Rodrigo A. R. Fischer, Electrical Engineering Department, University of Brasília, Brasília, Brazil, e-mail: rodrigoarfischer@gmail.com; João Paulo Leite, Electrical Engineering Department, University of Brasília, Brasília, Brazil, e-mail: jpauloleite@unb.br.

II. The Simulator

In order to obtain a user friendly reconfigurable simulator, and also to enable fast deployment time, the Python programming language was chosen. Nevertheless, only base external libraries were used, such as *numpy*.

The main drawback of using Python is the simulation execution time. Since a large number of frames need to be processed in order to obtain accurate statistics, some optimization is desired. By far, the turbo decoder is the most computationally complex block in the whole chain. Therefore, this module was optimized using *Pythran* [3], which, with only a few directives, statically compiles the subset of scientific kernels from Python to an optimized C++ implementation.

III. Turbo Coding

The proposed turbo scheme in DVB-RCS2 is composed of two parallel-concatenated double-bynary 16-state cyclic recursive systematic convolutional (CRSC) codes. Double-binary codes were introduced in [4], and perform significantly better than the conventional binary codes. Using the double-binary scheme offers a good trade-off of performance and computational complexity. The performance loss caused by the use of Max-Log-MAP Algorithm (a computationally less intensive version of the MAP decoding algorithm [5]) is almost non existent for double-binary codes [3].

A. Encoding

The encoding scheme is shown in Figure 1. The encoder core is a 16-state CRSC encoder. On the turbo encoding scheme, first a set of parity bits are generated, then the information is permuted, so that another set of parity bits can be generated. This results on a rate-$1/3$ code. To match the desired rate, a puncturing pattern is applied on the parity bits. The full description of the encoder can be found on [6].

B. Decoding

The decoding of turbo codes is done iteratively, as shown in Figure 2. The idea is that after being decoded by a CRCS decoder, the output likelihood has an extrinsic gain provided by the parity bits. This gain is inserted on the second CRCS, which then provides another gain. The subtraction operation is to make sure that the extrinsic information obtained by a CRSC isn't fed on itself, preventing the positive feedback of previously resolved information [7]. One iteration corresponds to the information passing both decoders. After the first

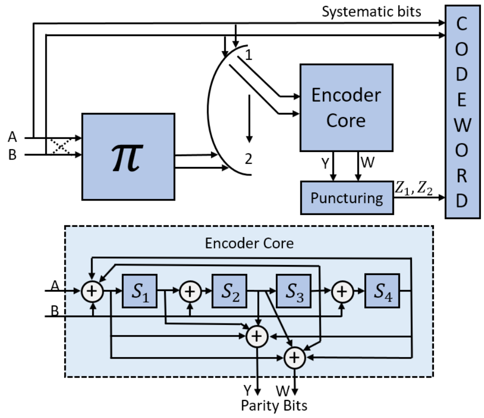

Fig. 1. DVB-RCS2 turbo encoder block diagram.

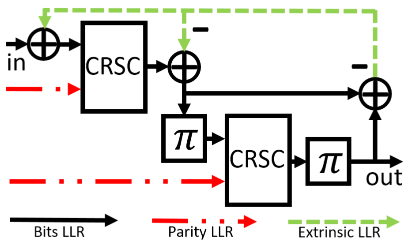

Fig. 2. DVB-RCS2 turbo decoder block diagram.

iteration, the extrinsic information obtained from the second decoder is summed with the first decoder input LLR's.

The Max-Log-MAP algorithm also introduces an overestimation of the extrinsic LLR's, which requires the extrinsic information to be multiplied by a scaling factor on every but the last one iteration [4]. The best observed scaling factor on the reference [4] was 0.7, and shall be adopted on this work.

IV. Results

The performance curves, shown in Figures 3 and 4, containing the packet error rate (PER) of a system composed of the whole DVB-RCS2 transmission-reception (TX-RX) chain with AWGN channel were obtained for the 28 waveform ID's (WF-ID) whose performances are in [2]. The packet sizes are the same as provided by [6], varying from 38 to 599 bytes, and coincide with the burst sizes. The performance points provided by [2] are shown as dots at PER $= 10^{-3}$.

V. Conclusion

A Python simulator was developed to asses the performance of the DVB-RCS2 satellite communications standard. The obtained performance has matching points in accordance with the standard, thus validating the implemented TX-RX processing chain. For a further work, channel models such as Rayleigh fading or Rice fading could be implemented to asses the impact of the propagation channel on the performance of the system.

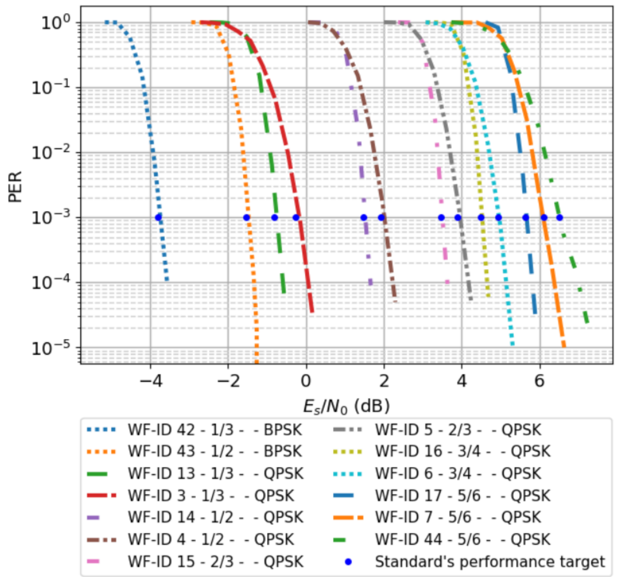

Fig. 3. DVB-RCS2 PER receiver performance for BPSK and QPSK modulations.

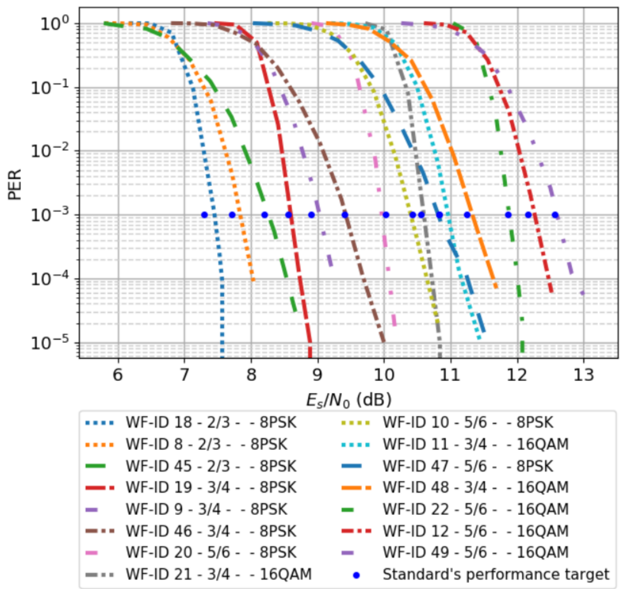

Fig. 4. DVB-RCS2 PER receiver performance for 8PSK and 16QAM modulations.

References

[1] Richharia, M., "Satellite Radio Interface Standards," in *Mobile Satellite Communications*: Principles and Trends, 2nd ed. West Sussex, United Kingdom: Wiley, 2014, ch. 8, sec. 3, pp. 407-411.

[2] *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 4: Guidelines for Implementation and Use of EN 301 545-2*, ETSI Standard TR 101 545-4, Apr. 2014.

[3] S. Guelton, P. Brunet, M. Amini, A. Merlini, X. Corbillon, A. Raynaud, "Pythran: Enabling static optimization of scientific Python programs," *Computational Science & Discovery*, vol. 8, no. 1, p. 014001, 2015.

[4] C. Douillard and C. Berrou, "Turbo codes with rate-m/(m+1) constituent convolutional codes," in IEEE Transactions on Communications, vol. 53, no. 10, pp. 1630-1638, Oct. 2005.

[5] H. R. Sadjadpour, R. Sonalkar and G. Jin "Proposal on using Multi-Tone Turbo Trellis Coded Modulation", T1E1.4 Meeting in Napa, CA, Oct. 2000.

[6] *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard*, ETSI Standard EN 301 545-1, Apr. 2014.

[7] Valenti, M., Cheng, S., Seshadri, R., "Turbo and LDPC Codes for Digital Video Broadcasting," in Turbo Code Applications, 1st ed., Netherlands: Springer, 2005, ch. 12, sec. 1, pp. 302-310.