IV
Simulation Scenarios

The experiment conducted in the present work employs a simplified simulation model trying to replicate the IP video environment, as close as possible, as far as packet sizes and channel erasure parameters are considered. The main target of this experiment is to compare the performance of video transmission under different erasure protection schemes — an LT based scheme is compared against two Reed-Solomon based schemes.

We made the choice of Reed-Solomon due to its known higher performance against simple parity codes. Reed-Solomon is also included as a possible scheme in the RFC2733 [4].

The environment including LT erasure protection is described in section IV.1 . The subsequent sections consider protecting the data with RS code based protection schemes.

With the built setup, the following will be the outcomes examined:

- Comparison of the overhead required by the scheme with the LT encoder to achieve video transmission with a given quality, with those required by the schemes which use Reed-Solomon codes to achieve the same quality;
- Comparison of the LT code and Reed-Solomon behaviors upon different channel erasure profiles;
- Real streaming of Transport Stream content and comparison of the result with the non-Fountain practical approach, when submitted through the same IP network.

The video quality will be quantified by using the objective Blocking Artifacts measurement presented in [12]. Subjective evaluation is also assessed by examining the decoded video content or printed frames included in the next chapter, which presents the results.

PUC-Rio - Certificação Digital Nº 0711234/CA

IV.1 Fountain Encoder Simulation

The Fountain encoder construction employed herein is shown in figure IV.1.

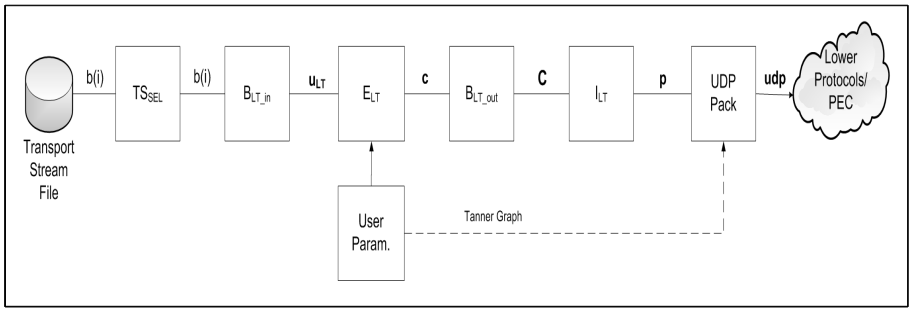

Figure IV.1: LT Encoder

PUC-Rio - Certificação Digital Nº 0711234/CA

Where the following notation is used:

b(i) is the individual byte read from the Transport Stream file.

$\mathbf{TS_{sel}}$ is the sync evaluation block of each TSP.

$\mathbf{B_{LTin}}$ is the LT encoder input buffer.

$\mathbf{u_{LT}}$ is the sequence of input blocks for the LT encoder.

$\mathbf{E_{LT}}$ is the LT encoder.

c is the sequence of encoded blocks from the LT encoder.

$\mathbf{B_{LTout}}$ is the encoder output buffer.

$\mathbf{I_{LT}}$ is the LT interleaver.

p is the sequence of UDP payload blocks.

$\mathbf{UDP_{pack}}$ is the UDP packetizer.

udp is the sequence of network packets.

Most relevant items will be explained in details in the following subsections.

(a) Input file and TS Selection Block (TS_{SEL})

The Transport Stream samples employed in this work can be found in [15]. Some samples do not start exactly with a sync byte, i.e. these do not match with the starting point of a TSP being carried.

The `TS Selection block` analyzes the syc bytes of the TSP's present in the TS sample and makes the selection of valid TSP's to be fed to the FEC input buffer.

In order to define the starting point of the Transport stream file being read as the first byte of the next integral Transport Stream packet, an auxiliary 188 bytes long vector entitled **a** is created. If the original TS file is valid, there will be a sync byte at some point in vector **a**. The vector entitled `input_aux` is the result of reading bytes directly from the source TS file:

```
fid=fopen('SPTS sample.ts','r');
input_aux=fread(fid);
fclose(fid);
a=input_aux(1:188);
```

The `start` variable indicates the position of the next sync byte found in the string, given by 47 HEX ($= 71DEC$).

```
start=find(a==71);
```

Finally, the input vector containing complete TSP's, denominated `input_TS`, is obtained:

```
input_TS=input_aux(start:((start-1)+(nTSPs*188)));
```

This pre-processing assures working with full Transport Stream packets in the original file. This operation can be repeated for every new TSP read, but it is expected that `start` will match the first byte position of every new TSP from this point on.

PUC-Rio - Certificação Digital Nº 0711234/CA

(b) Input Buffer (B_{LTin})

This block stores the incoming bytes read from the source file into vectors designated $\mathbf{u}_{\mathrm{LT}}(i)$, of length k, which corresponds to the LT encoder source block dimension.

Each vector $\mathbf{u}_{\mathrm{LT}}(i)$ is a sequence of TSP's, represented by vectors $\big(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{N_{\mathbf{tsp}}}\big)$.

Thus, the input buffer provides the sequence of LT source blocks

$$\mathbf{u}_{\mathrm{LT}} = \big(\mathbf{u}_{\mathrm{LT}}(1), \mathbf{u}_{\mathrm{LT}}(2), \ldots, \mathbf{u}_{\mathrm{LT}}(\beta)\big) \tag{1}$$

where β is the amount of LT source packets produced within the interval examined.

(c) LT Encoder block (E_{LT})

LT-encoding can be achieved by simply multiplying every component vector $\mathbf{u}_{\mathrm{LT}}(i)$ belonging to the sequence of vectors $\mathbf{u}_{\mathrm{LT}}$ by the matrix $\mathbf{G}$, of dimensions (k, n), which characterizes the LT-encoder. The encoder output is a sequence of vectors of length n, given by $\mathbf{c} = \big(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_\beta\big)$, where β is the amount of encoded blocks produced within the interval examined and each component vector is given by the notation

$$\mathbf{c}_i = \big(c_i(1), c_i(2), \ldots, c_i(n)\big). \tag{2}$$

User defined parameters

Still referring to fig. IV.1, the `User Param.` box will provide the set of parameters defined by the user to generate the matrix G employed for encoding.

The following parameters characterize the encoder:

δ is the failure probability defined by the user.

c is a constant defined by the user.

$\tau(i)$ is given by

$$\tau(i) = \begin{cases} \frac{R}{ik}, & \text{for } i = 1, ..., k/(R-1); \\ \frac{R}{k}\ln(\frac{R}{\delta}), & \text{for } i = k/R; \\ 0, & \text{for } i = k/R+1, ..., k; \end{cases}$$

$\beta(i)$ given by

$$\beta(i) = \sum_{i=1}^{k} \tau(i) + \rho(i);$$

PUC-Rio - Certificação Digital Nº 0711234/CA

$\mu(i)$ given by

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}$$

Once the Robust Soliton distribution $\boldsymbol{\mu} = \big(\mu(1), \ldots, \mu(i), \ldots, \mu(k)\big)$ is obtained, it will be used in the generation of the degrees vector $\mathbf{d}$, necessary for the construction of the generator matrix $\mathbf{G}$. The Robust Soliton Distribution $\boldsymbol{\mu}$ defines the probability of an element of the degree's vector $\mathbf{d}$ to assume an integer value between 1 and k.

In order to translate this information into the degrees vector $\mathbf{d}$, we make use of an auxiliary vector $\mathbf{T}$ of the same dimension of the vector $\mathbf{d}$:

$$\mathbf{T} = (T(1), \ldots, T(k+1)) \tag{3}$$

where:

$$\begin{aligned} T(1) &= 0 \\ T(i) &= \sum_{j=1}^{i-1} \mu(j), i = 2, \ldots, k+1. \end{aligned}$$

The vector $\mathbf{T}$ will define k intervals between $[0, 1]$, having threshold $T(i)$, for $i = 1 : k$. A uniform distribution in the interval $[0, 1]$ is sampled for $\ell = 1, \ldots, k$ and the degree $d(\ell)$ will be set equal to i if the sample value $d_s(i)$ falls within the interval $[T(i), T(i = 1)]$.

The generator matrix is created as a sparse matrix capable of allocating a number of non-zero elements equivalent to the sum of the elements of the degrees vector, from which G is generated. The n-th column of G is created with the amount of non-zero elements equivalent to the value of the n-th element in the degree vector generating the matrix.

Finally, a random permutation of the elements in the column of $\mathbf{G}$ assures that d source symbols are randomly chosen in the generation of each encoding symbol.

(d) Encoder's Output buffer (B_{LTout})

In the output buffer, each vector $c(i)$ of the sequence $\mathbf{c}$, provided by the encoder, is stored as the columns of a matrix entitled $\mathbf{C}$, of dimension $[n, \beta]$, where n is the length of the LT encoded block and β the amount of encoded blocks generated within the time interval observed and arranged as the columns of $\mathbf{C}$. The lower threshold for the output buffer capacity is then given by $n \cdot \beta$.

For notation purpose, $\mathbf{C}$ is given by:

PUC-Rio - Certificação Digital Nº 0711234/CA

$$\mathbf{C} = \big(c_1; c_2; \ldots, c_\beta\big). \tag{4}$$

(e) Interleaving (I_{LT}) and UDP packing (UDP_{pack})

The interleaver provides the sequence $\mathbf{p}$ from the matrix $\mathbf{C}$, by selecting groups of its lines. We picked a value of β such that an integer amount of lines is picked in order to compose one network packet.

$$\mathbf{p} = \big(p_1, p_2, \ldots, p_\varsigma\big). \tag{5}$$

where ζ is the amount of $\mathbf{p_i}$ blocks generated within the interval observed.

Each component $\mathbf{p_i}$ is given by the concatenation of ν lines of $\mathbf{C}$:

$$\mathbf{p_i} = \big(C(i,:), C(i+1,:), \ldots, C(:, i+(\nu-1)) \tag{6}$$

Hence, each vector $\mathbf{p_i}$ has size $\nu \cdot \beta$ bytes, being n the LT encoder output block size.

As a result, the interleaving operation can be characterized as follows:

$$\mathbf{p_i} = \big(c_1(i), c_2(i), \ldots, c_\beta(i), \ldots, c_\beta(i+(\nu-1))$$

where a zig-zag scan is accomplished across ν lines of the matrix $\mathbf{C}$. In other words, each network packet $\mathbf{udp_i}$ will contain ν lines of $\mathbf{C}$.

For the sake of simplification, in the simulation herein, β is made of the size of a TSP. Considering the MTU of $1,500$ bytes, ν equals seven.

The user parameters employed in the LT encoder, namely the information provided by the Tanner Graph, have to be provided to the LT decoder. We assume that the same is appended unscrambled to the first bytes of the payload section of each UDP packet. This is, for example, the method employed in ETSI102034 [21] and in the RFC5052 [9].

(f) IP Erasure Channel

In this box, we assume that the UDP packets are encapsulated into lower level protocols. Moreover, channel packet erasures occur, resulting in the random erasure of some components $\mathbf{UDP}(i)$ belonging to the sequence $\mathbf{UDP}$, depending on the channel erasure rate.

At the receiving end, a sequence

$$\mathbf{y} = \big(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N''}\big)$$

where $N'' \leq N'$, to account for packet losses. If no packets were lost, we will have $\mathbf{y}_i = \mathbf{x}_i$, otherwise $\mathbf{y}_i = \mathbf{x}_{\ell_i}$, with ℓ_i known to the receiver.

PUC-Rio - Certificação Digital Nº 0711234/CA

At implementation level, this can be easily accomplished by randomly erasing the columns in the generator matrix used to decode the incoming string, i.e. columns common to the same IP packet. It can be verified that erasing all none-zero elements in a column of the Generator matrix is the same as erasing all connections in the Tanner graph to a particular encoding symbol. This symbol will be handled as *unknown* by the binary erasure channel receiver.

IV.2 Fountain decoder simulation

The construction of the Fountain decoder implemented herein is depicted in figure IV.2.

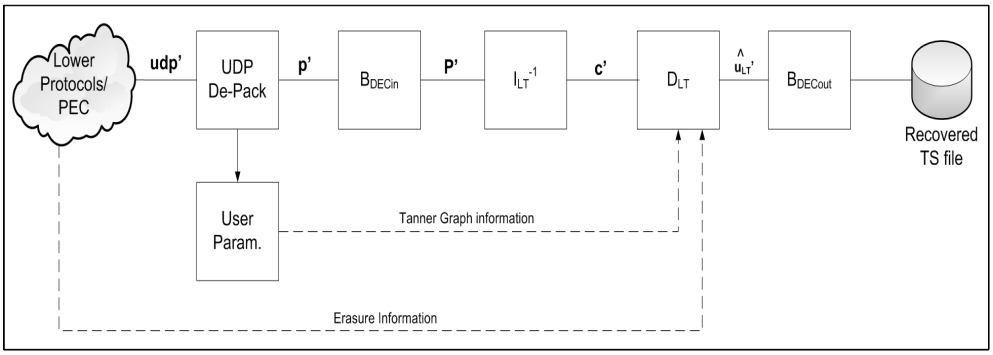

Figure IV.2: Fountain Decoder Simulation

The following notation is adopted:

udp' is the sequence of UDP packets entering the decoder.

UDP De-Pack is the de-packetizer that extracts payload information from incoming UDP packets.

p' is the sequence of received network packets' payload blocks.

$\mathbf{B_{DECin}}$ is the decoder's input buffer.

$\mathbf{I_{LT}^{-1}}$ is the decoder's inverse interleaver.

$\mathbf{c'}$ is the sequence of received LT encoded blocks.

$\mathbf{D_{LT}}$ is the LT decoder.

PUC-Rio - Certificação Digital Nº 0711234/CA

$\mathbf{u'_{LT}}$ is the sequence of recovered LT source blocks.

$\mathbf{B_{DECout}}$ is the decoder output buffer.

Most important items will be explained in details in the following sub-sections.

(a) UDP Unpacking block

This block receives the sequence $\mathbf{udp'} = (\mathbf{udp}_1, \mathbf{udp}_2, \ldots, \mathbf{udp}_\alpha)$, where α is the amount of received UDP packets and ς corresponds to the amount of UDP packets transmitted. Notice that due to erasures in the PEC channel, $\alpha \leq \varsigma$.

As the payload bytes $p(i)$ are unpacked, these are provided to the downstream buffer. We assume there is a mechanism to monitor packet loss, such as RTP sequence numbering in each RTP packet. Such mechanism can provide information to the upper layer channel decoder.

We also assume that this block retrieves information on how data was encoded at the transmitter side and forwards it to the downstream LT decoder.

(b) LT decoder input buffer (B_{DECin}) and de-interleaver (I_{LT}^{-1})

This block restores the payload sequences $\mathbf{p'}$ as ν rows of the matrix C'. Information for β and ν employed at the encoding side is necessary at this point. The parameters β and ν are explained in the previous sub-section.

Notice that:

```
size(C',1) <= size(C,1)
size(C',2) = size(C,2)
```

where the row sizes of matrices $\mathbf{C'}$ and $\mathbf{C}$ differ by a factor of ν.
The lines o $\mathbf{C'}$ are picked by I_{LT}^{-1} in order to compose the sequence $\mathbf{c'}$.
Each component of $\mathbf{c'_i}$ is given by:

$$\mathbf{c'_i} = (C'(:, i)) \tag{7}$$

The de-interleaving is characterized as hereunder:

$$\mathbf{c'_i} = (C'(1, i)), C'(2, i), \ldots, C'(n, i) \tag{8}$$

PUC-Rio - Certificação Digital Nº 0711234/CA

(c) LT Decoding

The `LT Decoder` block loads the Generator matrix G used in the encoding side. We assume the same is provided by the `UDP De-Pack` block.

It also reads the incoming recovered LT-encoded packets. We assume that the information concerning erasure positions is provided by lower level protocols. This information can be retrieved through RTP packet numbering, as explained in [1].

With the known erasure positions, the decoder erases all non-zero elements in the columns of $\mathbf{G}$, which correspond to the positions of erased packets.

The block's input is then given by the sequence of vectors (codewords) $\hat{\mathbf{c}}$. The sequence $\mathbf{u'}_{\mathrm{LT}} = \left(\mathbf{u'}_{\mathrm{LT}}(1), \mathbf{u'}_{\mathrm{LT}}(2), \ldots, \mathbf{u'}_{\mathrm{LT}}(K)\right)$, is provided by the decoder as a result of the decoding process — it should be kept in mind that for most symbols it will happen that:

$$\mathbf{u'}_{\mathrm{LT}}(j) = \mathbf{u}_{\mathrm{LT}}(j).$$

But if decoding does not achieve completion, it may also happen that, some symbols remain uncovered and delivered as erasure at the decoder output:

$$\mathbf{u'}_{\mathrm{LT}}(j) = E.$$

If such is the case, the erasure will be, arbitrarily, transformed into a zero (or a one) before being delivered to the the next processing box.

A brief explanation on the decoder's implementation is next ensued.

LT decoder implementation

First, a variable designated `Position` is created, given by the amount of non-zero elements in each column of `G`:

```
For i=1,...,N;
Position(i)= find(G(:,i));
end
```

When the decoder finds a column with a single non-zero element, it discovers the source symbol connected to that transmission node:

```
If Position(i)\equiv1
u_hat_LT(j) = u_LT(j);
end
```

Once the source symbol `u_hat_LT(j)` is discovered, it is summed modulo 2 to all transmission nodes – also known as *Neighbors* nodes – connected to it.

PUC-Rio - Certificação Digital Nº 0711234/CA

The positions of the *Neighbors* are defined by locating the non-zero elements in the `i`-th line of the generator matrix `G`, where `i` is defined in the previous step. We have thus: `Neighbors_Positions=find(G(Positions,:))`

The discovered source symbol is XOR'ed to the transmitted neighbors:

```
c_hat_in(Neighbors_Positions) = ...
    bitxor( c_hat_LT(Neighbors_Positions), u_hat_LT(j)(j))
```

Finally, the corresponding connections in Tanner graph are erased, what is the same as erasing the corresponding non-zero elements in `G`:

```
For Neighbors_Positions = 1:
G(j,Neighbors_Positions)=G(j,Neighbors Positions)-1
```

Fig. IV.3 illustrates the referred processing.

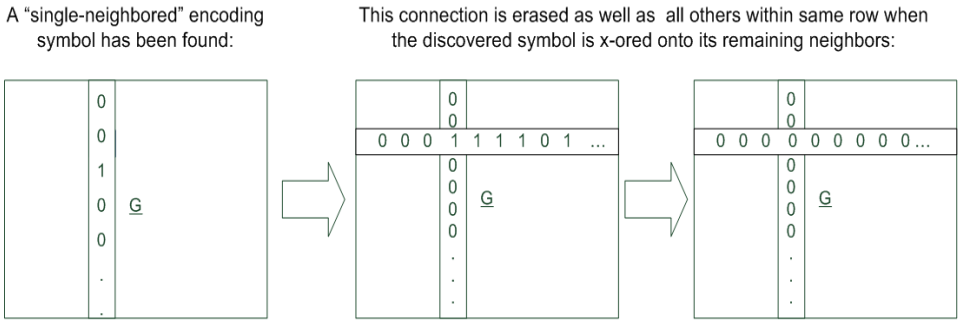

Figure IV.3: Processing of G upon each iteration

(d) LT decoder output buffer ($B_{LTDECout}$) and de-interleaving (I_{LT}^{-1})

The output buffer stores the decoded packets provided by `D_LT` as rows of a matrix with dimension $[L_{TS}, K_{LT}]$, both explained in previous sections. Finally, the de-interleaver recovers the original TSP's, which are written to the recovered file. Once all TSP's have been written, the same will be ready for analysis.

PUC-Rio - Certificação Digital Nº 0711234/CA

IV.3 Repeating the Experiment with Reed-Solomon Codes: one-dimensional scheme

The simulation presented in the previous section, in which the TS file is protected by an LT code, is repeated, for the sake of comparison, with schemes based on Reed-Solomon codes (RS codes, in short). Reed-Solomon codes are cited, for instance, in the RFC presented in [4], that defines a payload format for FEC in RTP streams. Reed-Solomon Codes are also frequently adopted as an optional coding scheme by equipment manufacturers.

Next, a single dimensional RS code with a pre-interleaving stage is employed. It is followed by a simulation that includes the second dimension of the RS code. The simulations herein are based in the parametrization provided in [20] and in the framework described in [9].

(a) RS-1D encoder

The RS-1D encoder is depicted in fig IV.4 and is very similar to the LT scheme presented in the previous section.

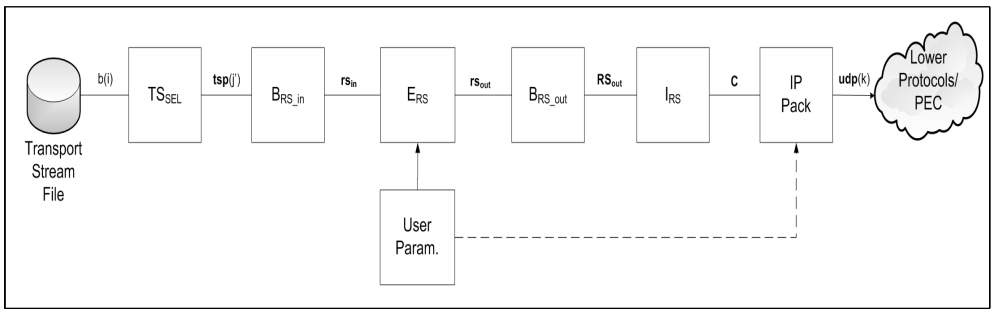

Figure IV.4: RS-1D Encoder

The following notation is employed:

b(i), $\mathbf{TS_{sel}}$, tsp(i), $\mathbf{IP_{pack}}$, UDP(j) have the same definition as in the LT simulation.

$\mathbf{B_{RSin}}$ is the RS encoder input buffer.

S is the matrix of TSPs stored in the input buffer.

$\mathbf{I_{RS}}$ is the RS interleaver.

PUC-Rio - Certificação Digital Nº 0711234/CA

$\mathbf{RS_{in}}$ is the RS source block.

$\mathbf{E_{RS}}$ is the RS encoder.

$\mathbf{RS_{out}}$ is the RS encoded block.

$\mathbf{B_{RSout}}$ is the Reed-Solomon IP packing buffer.

$\mathbf{C}$ is the matrix of RS encoded packets.

The composition of the UDP packs is very similar to IV.1. The matrix $\mathbf{C}$ has dimension $[L_{TSP}, n_{RS}]$, being n_{RS} the RS output block size. The packing into UDP packets is accomplished through selection of groups of seven columns of C, distributed along the payload section of each network packet. The simulation of packet erasures is the same.

(b) RS-1D decoder

The single-dimensional Reed-Solomon decoder is shown in figure IV.5.

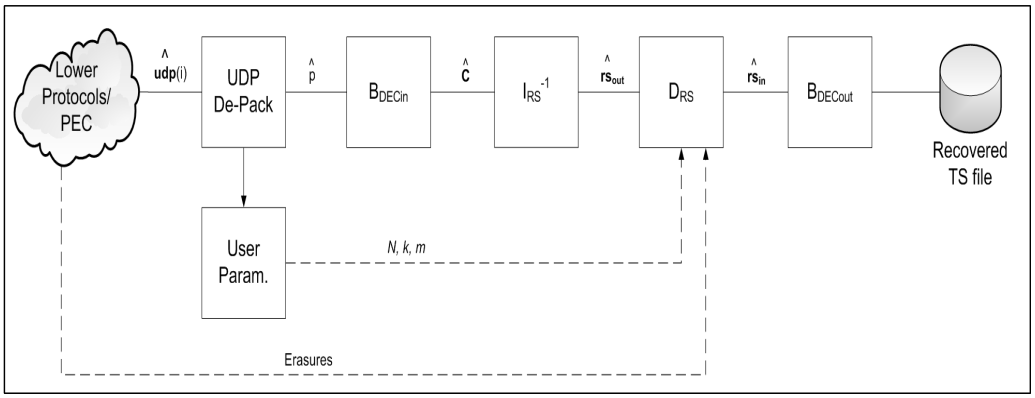

Figure IV.5: RS-1D decoder

The following notation is employed:

UDP'(i), IP de-pack,p'(j), tsp(k) are the same as in IV.1.

$\mathbf{B_{DECin}}$ is the RS decoder input buffer.

$\mathbf{RS'_{OUT}}$ is the RS encoded block received.

$\mathbf{D_{RS}}$ is the RS decoder.

$\mathbf{RS'_{IN}}$ is the recovered RS source block.

PUC-Rio - Certificação Digital Nº 0711234/CA

$\mathbf{B_{DECout}}$ is the RS decoder output buffer.

$\mathbf{I_{RS}^{-1}}$ is the inverse RS interleaver.

The RS Decoder box reads the incoming RS-encoded packets, the string of vectors (codewords) $\hat{\mathbf{c}}$. The sequence $\widehat{\mathbf{u}}_{\mathrm{RS}} = \big(\widehat{\mathbf{u}}_{\mathrm{RS}}(1), \widehat{\mathbf{u}}_{\mathrm{RS}}(2), \ldots, \widehat{\mathbf{u}}_{\mathrm{RS}}(K)\big)$, will be the sequence of symbols recovered by the decoder. Differently from the LT scheme, the decoder either accepts the packet as received without error or declares the whole packet as erased. In other words, it will either happen

$$\widehat{\mathbf{u}}_{\mathrm{LT}}(j) = \mathbf{u}_{\mathrm{LT}}(j).$$

or

$$\widehat{\mathbf{u}}_{\mathrm{LT}}(j) = E.$$

In such case, the erasures can be arbitrarily transformed into zeros (or into ones) before being delivered to the the next processing box.

PUC-Rio - Certificação Digital Nº 0711234/CA

IV.4 Repeating the Experiment with Reed-Solomon Codes: two-dimensional case

(a) RS-2D encoder

Fig. IV.6 shows the workflow for the two-dimensional Reed-Solomon encoder implemented herein.

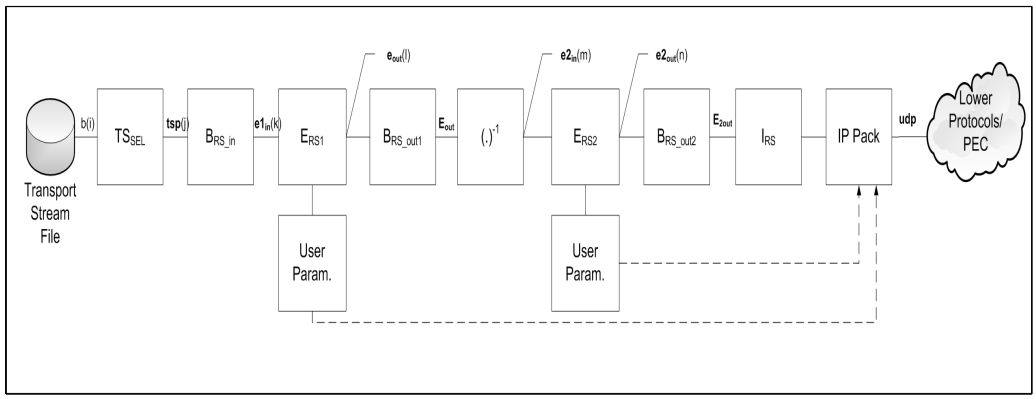

Figure IV.6: RS-2D Encoder

The following notation is employed:

b(i), $\mathbf{TS_{sel}}$, tsp(i), $\mathbf{B_{RSin}}$, S have the same definition as in the RS-1D scheme.

$\mathbf{I_{RS}}$ is the RS interleaver for the first dimension only.

$\mathbf{E_{RS1}}$, $\mathbf{E_{RS2}}$ are the RS encoders for the first and second dimensions, respectively.

$\mathbf{B_{RSout1}}$, $\mathbf{B_{RSout2}}$ are the output block packing buffers for the first and second dimensions, respectively.

$\mathbf{C_1}$,$\mathbf{C_2}$ are the matrices composed by output blocks of the first and second dimensions, respectively. These have sizes $[k_{RS1}, N_{RS1}]$ and $[k_{RS2}, N_{RS2}]$, being k_{RS1} and k_{RS2} the source block sizes and N_{RS1} and N_{RS2} output block sizes for the RS encoders of the first and second dimensions, respectively.

$\mathbf{RS_{PACK}}$ is the packetizer responsible for merging the original payload and both overheads of the two separate dimensions into the same matrix.

C is the matrix of size $[N_{RS1}, N_{RS2}]$, which contains both overheads from the two dimensions and the original payload bytes.

$\mathbf{IP_{PACK}}$ is the IP packing block, which distributes the bytes of **C** across the payload section of the UDP packets.

In the two-dimensional scheme, data is arranged into a long sequence of square matrices and is split between two paths. In the first, the same is inverted and line-wise encoded (vectors of length k_1 encoded with a systematic RS-encoder), whereas in the second path, no inversion is made and encoding is also performed on a line-by-line basis (vectors of length k_2 are encoded with an RS-encoder). The processing is illustrated in Fig. IV.7.

The overhead blocks generated by the second dimension of the code are indicated by *FEC'*. According to [20], the second dimension is intended to cope with single packet losses that might happen in addition to burst erasures.

The matrices C_1 and C_2 which result from storing the encoded blocks as its rows, are typically rectangular. The overheads have now to be merged with the original payload for transmission. These three components are arranged in such a way that the same UDP packet will not contain bytes from the original payload and from the resulting overheads, i.e., as if separate UDP packets were

PUC-Rio - Certificação Digital Nº 0711234/CA

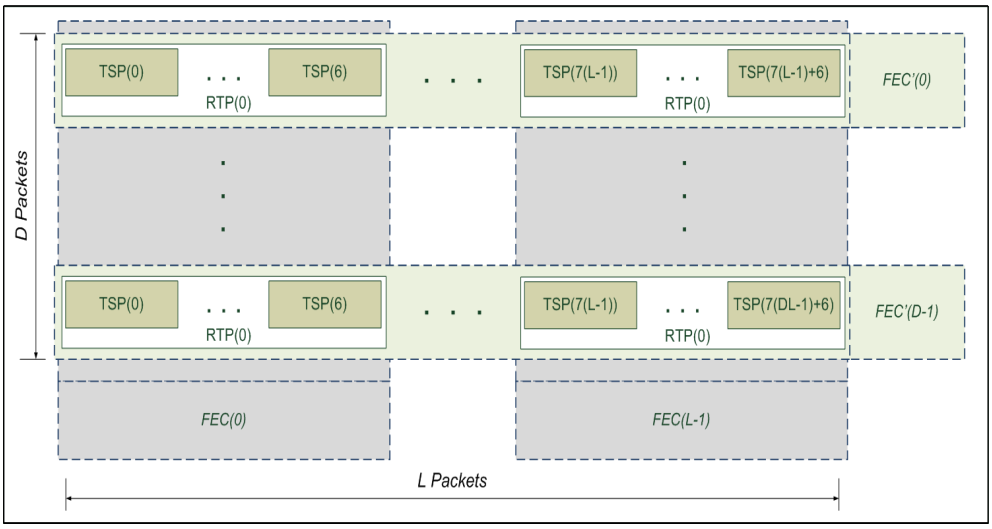

Figure IV.7: Two-Dimensional FEC

employed in the channel simulation, aiming to reproduce the framework in [21] or [4], where separate ports are employed.

The matrix **C**, shown in fig.IV.8, is composed as follows:

```
C = zeros(k,(2N-k));
C(:,1:N) = C_1(:,1:N);
C(:,N+1:(2N-k)) = C_1(:,k+1:N)
```

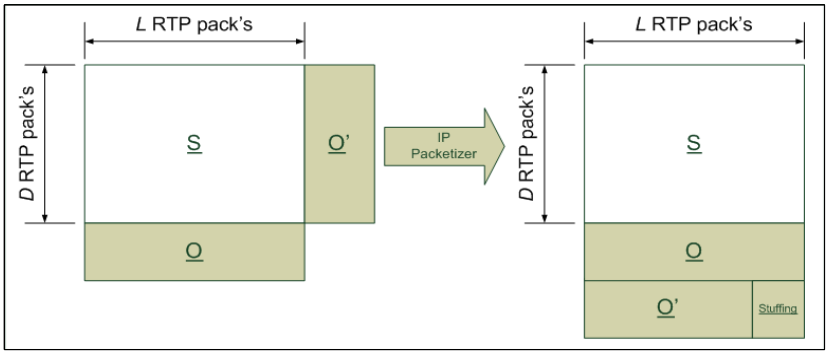

Figure IV.8: Re-arrangement of data subjected to transmission

PUC-Rio - Certificação Digital Nº 0711234/CA

In the `IP_Pack` block, the columns of **C** are distributed along the payload section of the UDP packets, as in the IP packing exposed in section IV.1.

Finally, we assume that the user parameters, necessary at the decoding side, are multiplexed in the `IP_Pack` block.

(b) RS-2D decoder

The RS-2D decoder is depicted in figure IV.9:

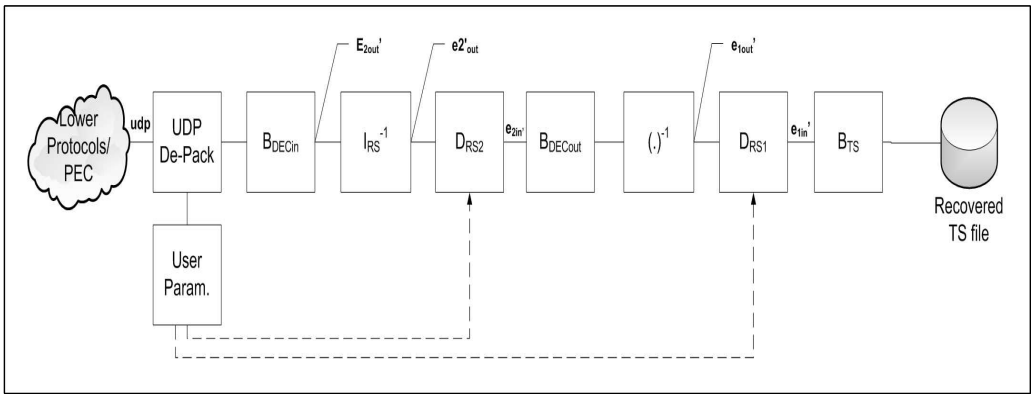

Figure IV.9: RS-2D decoder

The following notation is employed:

UDP'(i), IP de-pack,p'(j), tsp(k) are the same as in IV.1.

$\mathbf{B_{DECin}}$ is the RS-2D decoder input buffer.

$\mathbf{C'}$ is the RS-2D encoded block received.

$\mathbf{RS_{SEL1}}$ is the RS encoded packet selector for the first dimension.

$\mathbf{RS_{SEL2}}$ is the RS encoded packet selector for the second dimension.

$\mathbf{C'_1}, \mathbf{C'_2}$ are the recovered C_1 and C_2 matrices of payload and encoded packets, as defined in the previous section.

$\mathbf{D_{RS1}}$ is the RS decoder of the first dimension.

$\mathbf{D_{RS2}}$ is the RS decoder of the second dimension.

$\mathbf{RS'_{IN}}$ is the recovered RS source block.

PUC-Rio - Certificação Digital Nº 0711234/CA

$\mathbf{B_{DECout}}$ is the RS decoder output buffer.

$\mathbf{I_{RS}^{-1}}$ is the inverse RS interleaver.

The payload bytes de-capsulated from the UDP packets by the $B_{RSDECin}$ are arranged as lines of the matrix $\mathbf{C}$.

Again, we assume that the erasures have known positions, that were identified by RTP packet numbering and informed to upper layers of interest, such as the channel decoder. It can be verified that due to erasures in the IP channel:

```
size(C_recovered,2) >= size(C_,2)
```

The difference given by `size(C_recovered,2) - size(C_,2)` is a multiple of ξ, specified in the previous sub-section as the amount of columns grouped into the same UDP payload.

Hence, it can also be verified that:

```
size(C_1,2) >= size(C_1RX,2)
size(C_1,2) >= size(C_2RX,1)
```

C_1 and C_2 are provided by the blocks RS_{SEL1} and RS_{SEL2} respectively. These blocks literally "crop" the overhead column that is not useful for the corresponding downstream Reed-Solomon decoder.

The modules entitled `SUM` will check the integrity of `S'` provided by the first dimension.

PUC-Rio - Certificação Digital Nº 0711234/CA