A
Fountain Codes

Fountain Codes are Sparse Graph Erasure Codes introduced in [8], in the form of LT Codes its first practical realization, and are expected to work well with erasure channels. The transport of multimedia content over packet switching networks with erasures such as IP, can be considered as a target application for this type of codes.

This Appendix starts by presenting the main properties of Fountain Codes that make it attractive for IP video applications. Next, the encoding and decoding algorithms and main parameters are explained.

The aim is to assist with the understanding of the LT simulations presented in Chapter IV.

A.1 Main Properties

The main properties of Fountain codes include (1) the possibility of implementation of a rateless transmission scheme, where the sender does not need to know about specific channel details prior to transmission, (2) the *Universality property* and as consequence of the latter, (3) the benefits in Multicast scenarios.

(a) Rateless and Universality Properties

As mentioned, Fountain Codes can be classified as *rateless*. This results from the fact that Fountain codes have finite dimension and infinite block length, i.e., the amount of encoding symbols that can be generated from the same input block can be scaled up to infinite. It provides the ability of varying the code's rate for every encoded block during transmission and thus the *rateless* designation.

Therefore, Fountain codes are pointed as an interesting solution for cases where the channel conditions are unknown prior to encoding and the code's rate is adapted on the fly. In [17], such scenario is depicted and the use of a minimal acknowledgement is suggested, where the receiver transmits a termination bit to the encoder when the input block is successfully recovered.

The second aspect of interest regarding the Fountain coding scheme concerns the *Universality Property.* A Fountain code with dimension k is said Universal if it needs any subset of $k.(1/(1-p)+\varepsilon)$ encoded symbols for successfully decoding the input block with a probability that is at most $1/kc$, where ε is the overhead and p is the erasure probability of the BEC channel.

(b) Multicast scenario

The Universality Property makes Fountain codes very attractive for *Carousel Multicast* scenarios.

An explanation for this transmission mode is found in [14]. This scheme is commonly employed for distribution of media per context- or location-aware demands. As an example, clients connecting to an airport's mobile network (location-aware) and having its preferences displayed or connecting to a particular service (context-aware) within a multicast group.

The first challenge in this scenario is that receivers may be joining the carousel asynchronously. A new receiver that is joining the multicast group later will receive content reliably from that point in time on and not from the beginning of the multicast transfer. Moreover, in a network where reception conditions are diverse depending on the location, one user might not receive packets which were properly received by its neighbor. In such events, the client might request retransmission and as a result, other participants of the multicast group, that already received the same data reliably, will suffer with the retransmission delays and decrease of available bandwidth.

As an attempt to avoid the bandwidth waste and timing impairments, transmitters in Carousel Multicast mode commonly ignore retransmission requests when these are greater than a predefined threshold, what accounts for clients that joined the group at a later moment.

Another work around for retransmissions in multicast environments, is the distribution of the same information or portions of it through parallel paths. This type of scheme is denominated *Resilient Multicast*, explained in [25]. and [16].

Both solutions cited are not ideal, since redundant information will be transmitted. Universal Fountain codes are expected to cope with the reduction of retransmissions in a carousel scheme, even when the packets drop rate would not allow for successful decoding of the entire source block. This results from the fact that any new encoded symbol that arrives integral to the decoder will contribute with the decoding process. Hence, the client does not need to request retransmission, it simply waits for the next encoding packet that arrives without corruption, which will contribute equally for decoding success.

A.2 LT Encoding

The input of the LT encoder is defined by the source symbols vector $\bar{s} = \{s_1, s_2, ..., s_k\}$ with dimension k and its output by the encoding symbols vector $\bar{t} = \{t_1, t_2, ..., t_n\}$ with dimension N. The code's rate is defined by $R = k/N$.

Each encoding symbol, also known as transmission node $t(i)$ is obtained from XOR operation of the source symbols $s(i)$ or as a direct copy of it. Each $t(i)$ will have an associated degree $d(i)$, which is sampled from a degree distribution function and indicates how many source symbols are XOR'ed in order to compose the resulting encoding symbol. The degree distribution that will be sampled in order to define the degree of each encoded symbol is the key to the code's success.

The following steps describe the encoding algorithm for generation of the encoding symbol $t(i)$:

- Sample the degree distribution function to obtain the degree $d(i)$ of of the encoding symbol $t(i)$;
- Choose $d(i)$ source symbols at random;
- Perform the XOR operation of the source symbols chosen in the previous step, obtaining $t(i)$.

The operation is repeated until N encoded symbols are generated and the vector $\bar{t}$ is completed.

We can describe the generation of the Fountain encoded block as the multiplication of the source block by a Generator matrix G. The n^{th} transmission node t_n is given by the XOR operation, or bitwise sum modulo 2, of the source packets for which G_{nk}, the element in the n^{th} line and k^{th} column of the Generator matrix equals one. The transmitted packet t_n is given by:

$$t_n = \sum_{k=1}^{K} s_k G_{kn} \tag{1}$$

A Fountain encoded block can be completely described by a Tanner Graph. The same indicates the connections of each source symbol to the transmission nodes, of which generation it is contributing. All source symbols have to be connected. The amount of connections to a particular encoding symbol defines its degree and might vary from a single connection (a degree-one transmission node) up to k connections, when all source symbols contribute to the generation of that transmission node.

Each encoding symbol is associated to a column in the Generator matrix. Thus, the Tanner graph indicates the positions of 'ones' in G. The Graph is said Sparse because the mean degree is significantly smaller than k.

A.3 LT Decoding

The decoding algorithm is described by the following steps:

- Find the next encoded symbol with degree-one;
- "Discover" the original source symbol by simply replacing its value with the value of the degree-one encoded symbol found;
- Perform the XOR operation of the discovered input symbol into its neighbors and remove the corresponding Tanner Graph connections;
- Repeat the first step until all source symbols have been discovered and successful decoding is achieved.

If at some point during decoding, there is no single-degree element left at the decoder, the process is interrupted and a failure is declared. Depending on the protocol implementation, the client might request more packets or not.

The task of finding a single-degree element can be accomplished by simply examining the columns of the generator matrix in order to find the next column with a single non-zero element.

(a) Degree Distribution Considerations

The choice of the appropriate degree distribution is key to the code's success. The degree distribution aims to keep the costs associated to encoding and decoding as low as possible while assuring maximum decodability of the incoming string at low overheads upon channel erasures.

The first goal is accomplished by providing the lowest mean degree possible. The second is translated into having at least one single-degree encoded symbol left after each iteration during the decoding process.

The Ideal Soliton Distribution, given in [8], addresses the desirable behavior of having a single degree-one elements after each iteration and is given by:

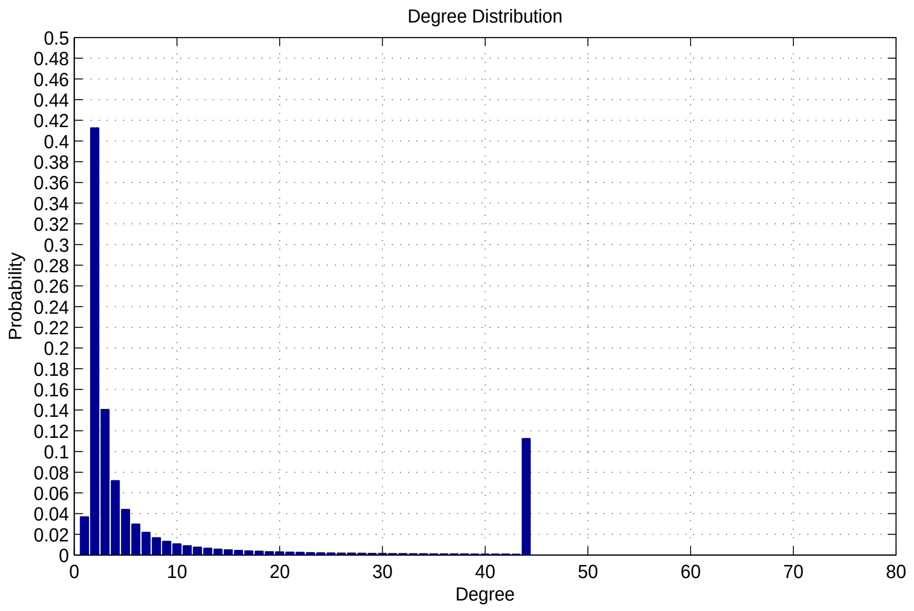

Figure A.1: Robust Soliton Distriution

$$\begin{cases} \rho(1) = 1/k, & \text{for d=1;} \\ \rho(d) = \frac{1}{d(d-1)}, & \text{for d=2, 3, ..., K.} \end{cases}$$

Still according to the same reference, the problem associated to this distribution is that the minimum fluctuation that occurs in practical decoding might cause the single degree-one element disappear at some point before full recovery of the original message, resulting in failure of the decoding process. In order to provide more robustness, a constant c and the failure probability were included, originating the *Robust Soliton Distribution*:

$$S \equiv c \ln(K/\delta)\sqrt{K}$$

This is the distribution employed in the simulations presented in Chapter IV. An example of such distribution is shown in figure A.1.

B
Adaptive Channel Encoder

B.1 Simulation results and background

By examining he results of the simulations presented in Chapter IV, it was noted that, for some decoded samples, the byte degradation was lower than others, but the playback application [24] was presenting a few time-outs or not even opening the decoded content.

The byte degradation measurement employed does not consider the upper Transport Stream layer. When checking Transport Stream alarms, explained in Chapter II, with the assistance of a TS analyzer [7], it was verified that whenever "time-outs" or the "non-decodeability" symptoms were observed, under low degradation at byte-level, PSI information was corrupted.

Hence, the scheme proposed herein provides greater robustness to the incoming stream whenever PSI information is found. In a latter approach, PCR TSP's were also included in this "privileged" information block, considering its importance, as reviewed in Chapter II.

One can think of other fields that could function as criteria for varying the rate of the Fountain channel encoder. As an example, in a Multi-program Transport Stream scenario, the service provider might want to privilege a particular service, considered more important. In such case, the flags cited hereunder, indicating the importance of the incoming TSP or the Elementary stream being carried by the same, could trigger the transmission of additional encoded symbols for that packet:

- the Elementary Stream Priority bit, the 10^{th} bit in the Adaptation Field of the TSP;
- the Transport Priority bit, the 10^{th} bit in the header of the TSP.

Since the present simulation is working with a single program Transport Stream, let us focus in the decoder synchronization only and make use of the Adaptation Field and PCR flags as criteria for triggering the transmission of additional symbols.

B.2 Overall workflow

The workflow for this approach is shown in figure B.1. At the encoder, when the Transport Stream file is read, a parallel block is analyzing the incoming TSPs. This is not intended to be a complete analysis, only of the fields of interest, at low processing costs.

For the selection of PCR information, the items analyzed, include the Adaptation Field presence flag, ($s_h(26)$) and $s_h(27)$ and the PCR presence flag, $s_h(26)$ and $s_a(11)$, explained in II. If both bits are set, the *TS Analysis block* will switch the incoming TSP to the appropriate channel encoder employing differentiated Generator matrices.

For PSI information, the examination first looks for the known PAT PID, where it finds the list of PMT's and from this point on, the same are included in the search.

Since it was observed, in the LT code performance presented in fig. B.1, that, when close to decoding completion, an increment of one or two percent in the overhead is capable of significantly improving the decode-ability up to the entire block, we made the choice of providing an additional four percent of bytes of the entire LT encoder input block upon detection of PCR or PSI, i.e., twice the overhead increment capable of assuring successful decoding.

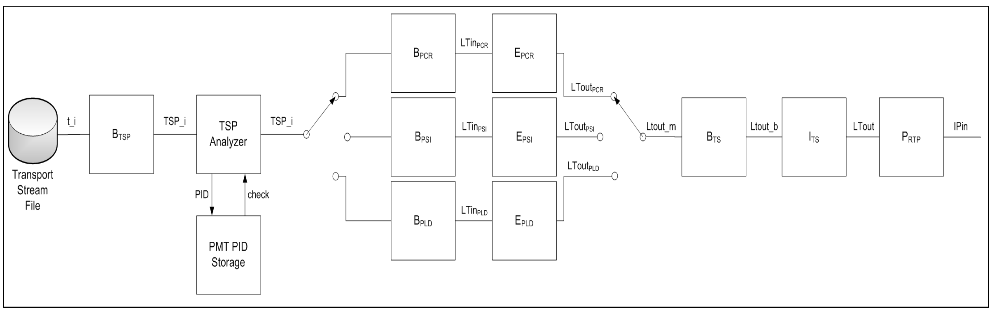

Figure B.1: Adaptive LT encoder considering PSI and PCR

The following notation is employed.

t(i) is the i^{th} bit entering the encoder, resulting from the reading process.

$\mathbf{B_{TSP}}$ is the Transport Stream packet buffer, of the size of one TSP of 188 Bytes.

TSP(j) is the j_{th} entire Transport Stream packet stored in B_{TSP} being fed to the analysis block.

TSP Analyzer this block will be checking for presence of PSI or PCR information in the TSP.

PMT PID storage This block will store the PMT pid for the single program – in this section we evaluate an SPTS – once the same is obtained from the PAT. It will be used for checking PMT presence in the upcoming packets.

$\mathbf{B_{PCR}}, \mathbf{B_{PSI}}, \mathbf{B_{PLD}}$ The PCR Buffer, PSI Buffer and Payload Bufer, respectively.

$\mathbf{LT_{PCR}}, \mathbf{LT_{PSI}}, \mathbf{LT_{PLD}}$ input packets to the LT encoding process.

$\mathbf{E_{PCR}}, \mathbf{E_{PSI}}, \mathbf{E_{PLD}}$ LT encoding of LT_{PCR},LT_{PSI} and LT_{PLD} respectively.

$\mathbf{LTout_{PCR}}, \mathbf{LTout_{PSI}}, \mathbf{LTout_{PLD}}$ output packets of E_{PCR},E_{PSI},E_{PLD} respectively.

$\mathbf{B_{TS}}$ is the Buffer for the string composed of LT_{PCR},LT_{PSI} and LT_{PLD}.

$\mathbf{I_{TS}}$ Interleaver for the string composed of LT_{PCR},LT_{PSI} and LT_{PLD}.

$\mathbf{P_{RTP}}$ Packetizer of the adaptive encoding scheme into RTP/UDP/IP packets.

(a) Preliminary Transport Stream Analysis Block

This block reads the original Transport Stream file and reshapes the same into a matrix which columns are given by vectors corresponding to the Transport Stream packets of length 188 Bytes each.

$$\mathbf{s} = \left(\mathbf{s}_1; \mathbf{s}_2; \ldots; \mathbf{s}_{N_{\mathbf{tsp}}}\right) \tag{1}$$

is now is a matrix of dimension $188 \times L$ and each vector component has the form

$$\mathbf{S} = \begin{pmatrix} s(i,1) \\ s(i,2) \\ \cdots \\ s(i,k_2) \end{pmatrix}. \tag{2}$$

This matrix of bytes has each of its elements converted into binary format, thus, the *Adaptation Field Control Flag* and the *PCR Presence Flag* are composed by two and one bits respectively.

After conversion into binary format, the matrix is inverted and reshaped into a new matrix named `M_bin` of dimension $[N_{\mathbf{tsp}}, Nlin]$, where $Nlin$ is clearly the amount of bits per TSP.

```
nTSPs=numel(S)/TSP_length
input_s=reshape(input_s,TSP_length,nTSPs);
bin=dec2bin(input_s);
S_bin=bin';
s_bin=reshape(M_bin,1504,nTSPs);
S_bin=double(M_bin);
```

As seen in chapter II, the PCR may be presented inside the Adaptation Field, thus the presence of the same is a necessary condition and its control flag needs to be verified as well.

The Adaptation field and the PCR positions across all TSPs can be verified through examination of lines 27 and 44 of `M_bin` respectively:

```
afield=input(27,:);
afpositions=find(afield==49)

pcr=input(44,:);
pcrpositions=find(pcr==49);
```

Since we are looking for columns in which both flags are set, the following condition is implemented:

```
PCR_Positions=find(pcr==49 & afield==49)
```

The elements of the vector `PCR_Positions` identify the TSPs carrying PCR. As mentioned, this information is forwarded to the *Generator matrix block*. As soon as the encoder reaches these portions of the stream, an input block N', of size $1.02N$ is employed.

```
G = spalloc(N',Ncol,nzmax)
```

In the simulations next ensued, values of N which are multiple integers of the TS packet size have been employed. This approach might require cropping of the original input file, more specifically of the first and last TSP's if the same are not integral. Another work around would be the addition of stuffing bytes at the end of the input block to the LT encoder. We chose the first option.

The identification of Program Map Table is performed as follows.

```
TSPn=input_symbol(:,n);
TSPn_HEX=dec2hex(TSPn)
b=dec2bin(TSPn)
PIDbin=b(3)
PIDhex=[TSPn_HEX(3);TSPn_HEX(3+188)]'
PIDdec=hex2dec(PIDhex)
```

The code selects the value of n that returns $PIDhex = 47$.

Once the same found, the PMT value is retrieved from the PMT. From this point on, PMT PID's will also serve as criteria for the analysis block.

B.3 Results

Referring to the LT Performance presented in figure V.1, the present scheme was applied to the point in the curve between overhead values of 1.075 and 1.085. At this point, few additional overhead will suffice to completely decode the LT packet. This was assured when an extra 4 percent was transmitted for the PCR TSP's. As a result, no PCR errors were found in the decoded string.

The result is visible when playing back the decoded content with VLC application [24]. In most simulations, the decoder was able to playback the entire content without interruption, even though visual impairment was observed in some portions of the decoded content. Employing a "flat" LT [8520,7896] as presented in section IV.1, significantly more time-outs and visual impairments were observed.

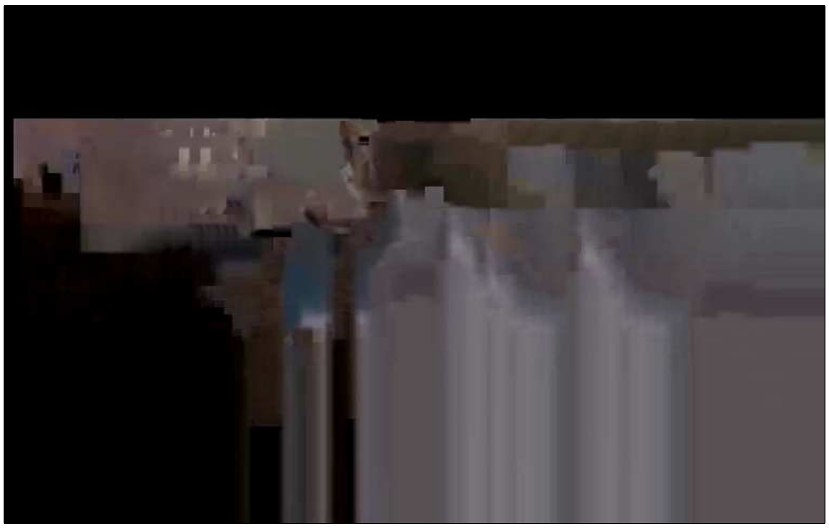

Figure B.2: Sample frame of "flat" LT at $N = 1.075k$

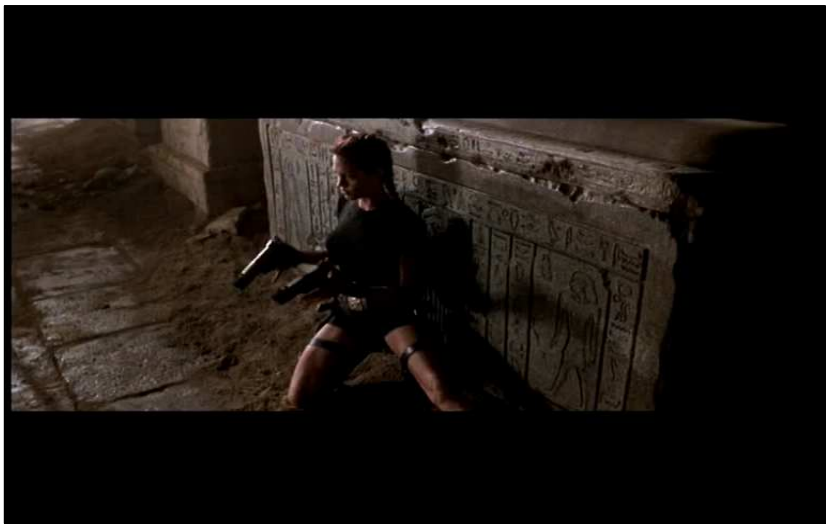

Figure B.3: Sample frame of "Adaptive" LT at N slightly larger than $1.075k$

Bibliography

[1] R. Frederick V. Jacobson H. Schulzrinne, S. Casner, *Rtp: A transport protocol for real-time applications*, RFC1889, January 1996. III.2(b), IV.2(c)

[2] Advanced Television Systems Committee Inc., *Recommended practice: Transport stream verification*, Document A/78A, 9 May 2007, 2001. I, II.2(b)

[3] ISO/IEC, *Information technology; generic coding of moving pictures and associated audio information*, 13818-1, 2007. 4, II.2(b)

[4] H. Schulzrinne J. Rosenberg, *An rtp payload format for generic forward error correction*, RFC2733 (1999). III.2(b), IV, IV.3, IV.4(a)

[5] J. Clark J. Welch, *A proposed media delivery index*, RFC-4445, 2006. I

[6] *Pcr measurement primer*, Tektronix, 2002. II.2(b)

[7] *Ts reader*, www.tsreader.com, 2010. B.1

[8] M. Luby, *Lt codes*, 2002. A, A.3(a)

[9] L. Vicisano M. Luby, M. Watson, *Forward error correction (fec) building block*, RFC5052, 2009. IV.1(e), IV.3

[10] P. A. Chou M. Van der Schar, *Multimedia over ip and wireless networks, compression, networking and system*, Elsevier Inc., 2007. III.1, III.2(a), V, V.2(c), VI

[11] T. K. Moon, *Error correction coding – mathematical methods and algorithms*, Wiley Interscience, 2005. III.2

[12] R. Muijs and I. Kirenko, *A no-reference blocking artifact measure for adaptive video processing*, Philips Research Laboratories, 1–4. I, IV, V.4

[13] K. Cornog N. Hurst, *Mpeg splicing*, SMPTE, 1997. 3

[14] I. Miloucheva N. Reyes, J. Mahnke, *Multicast retransmission strategies for content delivery in heterogeneous mobile ip environment*, 2001. A.1(b)

[15] Roku HD1000 Projects, *Sample hdtv transport streams*, http://www.dododge.net/roku/ts-samples.html, 2004. IV.1(a)

[16] B. Bhattacharjee A. Srinivasan S. Banerjee, S. Lee, *Resilient multicast using overlays*, Department of Computer Science of the Universityu of Maryland, 1–12. A.1(b)

[17] A. Shokrollahi, *Fountain codes*, Ecole Polytechnique Federale de Lausanne (2003), 1–4. A.1(a)

[18] *Subjective video quality assessment methods for multimedia applications*, ITU-T P.910, 1999. V.4

[19] *Methodology for the subjective assessment of the quality of television pictures*, ITU-R BT.500, 2000. V.4

[20] *Transmission of professinal mpeg-2 transport streams over ip networks*, Pro-MPEG FEC Code of Practice 3.2, 2004. III.2(b), IV.3, IV.4(a)

[21] *Digital video broadcasting; transport of mpeg-2 ts based dvb services over ip based networks*, ETSI TS 102 542 V1.2.1 (04/08), 2008. IV.1(e), IV.4(a)

[22] *A proposed time-stamped delay factor algorithm for measuring network jitter on rtp streams*, EBU TECH - 337, 2010. I

[23] European Broadcasting Union, *Digital video broadcasting; measurement guidelines for dvb systems*, ETSI ETR 101 290 V1.2.1 (2001-05), 2001. I, II.2(b)

[24] VideoLAN, *Vlc*, www.videolan.org, 2010. B.1, B.3

[25] C. T. Chou S. K. Jha X. Zhao, J. Guo, *Resilient multicasting in wireless mesh networks*, School of Computer Science and Engineering of the University of New South Wales (2004), 1–4. A.1(b)

[26] B. Zellner, *Comprehensive pcr analysis*, Tektronix Application Note, 2006. II.2(b)