Tetrys, an On-the-Fly Network Coding protocol
draft-detchart-nwcrg-tetrys-04

Abstract

   This document describes Tetrys, an On-The-Fly Network Coding (NC)
   protocol that can be used to transport delay and loss sensitive data
   over a lossy network. Tetrys can recover from erasures within a RTT-
   independent delay, thanks to the transmission of coded packets. It
   can be used for both unicast, multicast and anycast communications.

Status of This Memo

Copyright Notice

Table of Contents

1. **Introduction**

   This document describes Tetrys, a novel network coding protocol.
   Network codes were introduced in the early 2000s [AHL-00] to address
   the limitations of transmission over the Internet (delay, capacity
   and packet loss). While the use of network codes is fairly recent in
   the Internet community, the use of application layer erasure codes in
   the IETF has already been standardized in the RMT [RMT] and the
   FECFRAME [FECFRAME] working groups. The protocol presented here can
   be seen as a network coding extension to standards solutions. The

current proposal can be considered as a combination of network erasure coding and feedback mechanisms [Tetrys].

The main innovation of the Tetrys protocol is in the generation of coded packets from an elastic encoding window periodically updated with the receiver's feedbacks. This update is done in such a way that any source packets coming from an input flow is included in the encoding window as long as it is not acknowledged or the encoding window did not reach a size limit. This mechanism allows for losses on both the forward and return paths and in particular is resilient to acknowledgement losses.

With Tetrys, a coded packet is a linear combination over a finite field of the data source packets belonging to the coding window. The choice of the finite field of the coefficients is a trade-off between the best performance (with non-binary coefficients) and the system constraints (binary codes in an energy constrained environment) and is driven by the application.

Thanks to the elastic encoding window, the coded packets are built on-the-fly, by using an algorithm or a function to choose the coefficients. The redundancy ratio can be dynamically adjusted, and the coefficients can be generated in different ways along a transmission. Compared to FEC block codes, this allows to reduce the bandwidth use and the decoding delay.

1.1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Definitions, Notations and Abbreviations

The terminology used in this document is presented below. It is aligned with the FECFRAME terminology as well as with recent activities in the Network Coding Research Group.

Source symbol: a symbol that has to be transmitted between the ingress and egress of the network.

Coded symbol: a linear combination over a finite field of a set of source symbols.

Source symbol ID: a sequence number to identify the source symbols.

Coded symbol ID: a sequence number to identify the coded symbols.

Encoding coefficients: elements of the finite field characterizing the linear combination used to generate a coded symbols.

Encoding vector: set of the encoding coefficients and input source symbol IDs.

Source packet: a source packet contains a source symbol with its associated IDs.

Coded packet: a coded packet contains a coded symbol, the coded symbol's ID and encoding vector.

Input symbol: a symbol at the input of the Tetrys Encoding Building Block.

Output symbol: a symbol generated by the Tetrys Encoding Building Block. For a non systematic mode, all output symbols are coded symbols. For a systematic mode, output symbols can be the input symbols and a number of coded symbols that are linear combinations of the input symbols + the encoding vectors.

Feedback packet: a feedback packet is a packet containing information about the decoded or received source symbols. It can also bring additional information about the Packet Error Rate or the number of various packets in the receiver decoding window

Elastic Encoding Window: an encoder-side buffer that stores all the non-acknowledged source packets of the input flow that are involved in the coding process.

Coding Coefficient Generator Identifier: a unique identifier that define a function or an algorithm allowing to generate the encoding vector.

Code rate: Define the rate between the number of input symbols and the number of output symbols.

3. Architecture

-- Editor's note: The architecture used in this document should be aligned with the future NC Architecture document [NWCRG-ARCH]. --

3.1. Use Cases

Tetrys is well suited, but not limited to the use case where there is a single flow originated by a single source, with intra stream coding that takes place at a single encoding node. Note that the input stream can be a multiplex of several upper layer streams.

Transmission can be over a single path or multiple paths. In
addition, the flow can be sent in unicast, multicast, or anycast
mode. This is the simplest use-case, that is very much inline with
currently proposed scenarios for end-to-end streaming.

3.2. Overview

```
+----------+                                      +----------+
|          |                                      |          |
|   App    |                                      |   App    |
|          |                                      |          |
+----------+                                      +----------+
     |                                                 ^
     |   source                             source     |
     |   symbols                            symbols     |
     |                                                  |
     v                                                  |
+----------+              +----------+             +----------+
|          | output packets |        | output packets |       |
|  Tetrys  |--------------->|  Tetrys |--------------->| Tetrys |
|  Encoder |feedback packets| Recoder |feedback packets| Decoder |
|          |<---------------|         |<---------------|        |
+----------+              +----------+             +----------+
```
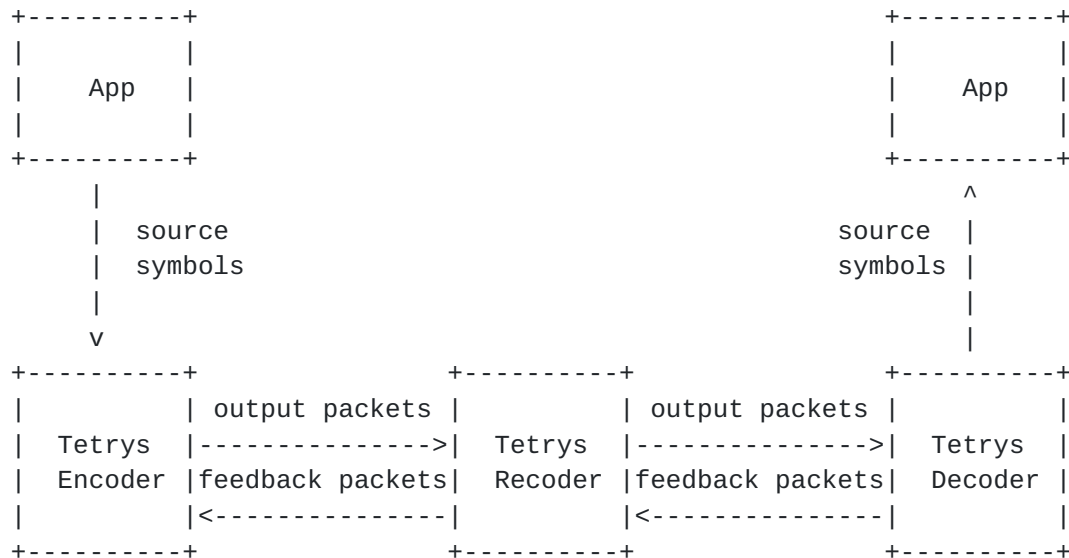
                    Figure 1: Tetrys Architecture

The Tetrys protocol features several key functionalities:

o On-the-fly encoding;

o Recoding;

o Decoding;

o Signaling, to carry in particular the symbol identifiers in the
   encoding window and the associated coding coefficients when
   meaningful, in a manner that was previously used in FEC;

o Feedback management;

o Elastic window management;

o Channel estimation;

o Dynamic adjustment of the code rate and flow control;

o Congestion control management (if appropriate);

-- Editor's note: must be discussed --

o Tetrys packet header creation and processing;

o -- Editor's note: something else? --

These functionalities are provided by several building blocks:

o The Tetrys Building Block: this BB is used during encoding,
   recoding and decoding processes. It must be noted that Tetrys
   does not mandate a specific building block. Instead any building
   block compatible with the elastic encoding window feature of
   Tetrys can be used.

o The Window Management Building Block: this building block is in
   charge of managing the encoding encoding window at a Tetrys
   sender.

      -- Editor's note: Is it worth moving it in a dedicated BB? To
      be discussed --

o Other ?

In order to enable future components and services to be added
dynamically, Tetrys adds a header extension mechanism, compatible
with that of LCT, NORM, FECFRAME [REFS].

4. Packet Format

4.1. Common Header Format

All types of Tetrys packets share the same common header format (see
Figure 2).

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   V   | C |S|      Reserved       |    HDR_LEN    |  Packet Type  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Congestion Control Information (CCI, length = 32*C bits)     |
 |                              ...                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Transport Session Identifier (TSI, length = 32*S bits)     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                  Header Extensions (if applicable)             |
 |                              ...                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                        Figure 2: Common Header Format

     -- Editor's note: this format inherits from the LCT header format
     (RFC 5651) with slight modifications. --

  o Tetrys version number (V): 4 bits. Indicates the Tetrys version
     number. The Tetrys version number for this specification is 1.

  o Congestion control flag (C): 2 bits. C=0 indicates the Congestion
     Control Information (CCI) field is 0 bits in length. C=1
     indicates the CCI field is 32 bits in length. C=2 indicates the
     CCI field is 64 bits in length. C=3 indicates the CCI field is 96
     bits in length.

        -- Editor's note: version number and congestion control to be
        discussed --

  o Transport Session Identifier flag (S): 1 bit. This is the number
     of full 32-bit words in the TSI field. The TSI field is 32*S bits
     in length, i.e., the length is either 0 bits or 32 bits.

  o Reserved (Resv): 9 bits. These bits are reserved. In this
     version of the specification, they MUST be set to zero by senders
     and MUST be ignored by receivers.

  o Header length (HDR_LEN): 8 bits. Total length of the Tetrys
     header in units of 32-bit words. The length of the Tetrys header
     MUST be a multiple of 32 bits. This field can be used to directly
     access the portion of the packet beyond the Tetrys header, i.e.,
     to the first other header if it exists, or to the packet payload
     if it exists and there is no other header, or to the end of the
     packet if there are no other headers or packet payload.

  o Packet Type: 8 bits. Type of packet.

o Congestion Control Information (CCI): 0, 32, 64, or 96 bits Used
   to carry congestion control information. For example, the
   congestion control information could include layer numbers,
   logical channel numbers, and sequence numbers. This field is
   opaque for the purpose of this specification. This field MUST be
   0 bits (absent) if C=0. This field MUST be 32 bits if C=1. This
   field MUST be 64 bits if C=2. This field MUST be 96 bits if C=3.

o Transport Session Identifier (TSI): 0 or 32 bits The TSI uniquely
   identifies a session among all sessions from a particular sender.
   The TSI is scoped by the IP address of the sender, and thus the IP
   address of the sender and the TSI together uniquely identify the
   session. Although a TSI in conjunction with the IP address of the
   sender always uniquely identifies a session, whether or not the
   TSI is included in the Tetrys header depends on what is used as
   the TSI value. If the underlying transport is UDP, then the
   16-bit UDP source port number MAY serve as the TSI for the
   session. If there is no underlying TSI provided by the network,
   transport or any other layer, then the TSI MUST be included in the
   Tetrys header.

4.1.1. Header Extensions

Header Extensions are used in Tetrys to accommodate optional header
fields that are not always used or have variable size. The presence
of Header Extensions can be inferred by the Tetrys header length
(HDR_LEN). If HDR_LEN is larger than the length of the standard
header, then the remaining header space is taken by Header
Extensions.

If present, Header Extensions MUST be processed to ensure that they
are recognized before performing any congestion control procedure or
otherwise accepting a packet. The default action for unrecognized
Header Extensions is to ignore them. This allows the future
introduction of backward-compatible enhancements to Tetrys without
changing the Tetrys version number. Non-backward-compatible Header
Extensions CANNOT be introduced without changing the Tetrys version
number.

There are two formats for Header Extensions, as depicted in Figure 3.
The first format is used for variable-length extensions, with Header
Extension Type (HET) values between 0 and 127. The second format is
used for fixed-length (one 32-bit word) extensions, using HET values
from 128 to 255.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   HET (<=127)  |       HEL       |                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                           +
.                                                               .
.               Header Extension Content (HEC)                 .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  HET (>=128)  |       Header Extension Content (HEC)         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
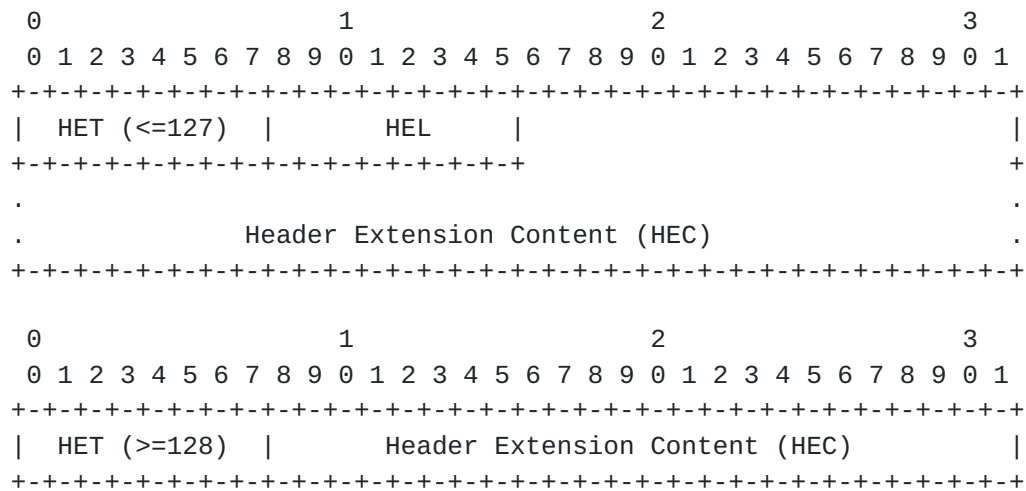
Figure 3: Header Extension Format

o Header Extension Type (HET): 8 bits The type of the Header
   Extension. This document defines a number of possible types.
   Additional types may be defined in future versions of this
   specification. HET values from 0 to 127 are used for variable-
   length Header Extensions. HET values from 128 to 255 are used for
   fixed-length 32-bit Header Extensions.

o Header Extension Length (HEL): 8 bits The length of the whole
   Header Extension field, expressed in multiples of 32-bit words.
   This field MUST be present for variable-length extensions (HETs
   between 0 and 127) and MUST NOT be present for fixed-length
   extensions (HETs between 128 and 255).

o Header Extension Content (HEC): variable length The content of the
   Header Extension. The format of this sub-field depends on the
   Header Extension Type. For fixed-length Header Extensions, the
   HEC is 24 bits. For variable-length Header Extensions, the HEC
   field has variable size, as specified by the HEL field. Note that
   the length of each Header Extension MUST be a multiple of 32 bits.
   Also note that the total size of the Tetrys header, including all
   Header Extensions and all optional header fields, cannot exceed
   255 32-bit words.

4.2. Source Packet Format

A source packet is the encapsulation of a Common Packet Header, a
Source Symbol ID and a source symbol (payload). The source symbols
can have variable sizes.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 /                     Common Packet Header                      /
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Source Symbol ID                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 /                            Payload                            /
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Source Packet Format

Common Packet Header: a common packet header (as common header format) where Packet Type=0.

Source Symbol ID: the sequence number to identify a source symbol.

Payload: the payload (source symbol)

4.3. Coded Packet Format

A coded packet is the encapsulation of a Common Packet Header, a Coded Symbol ID, the associated Encoding Vector and a coded symbol (payload). As the source symbols CAN have variable sizes, each source symbol size need to be encoded, and the result must be stored in the coded packet as the Encoded Payload Size (16 bits): as it is an optional field, the encoding vector MUST signal the use of variable source symbol sizes with the field V (see Section 6.1.1.2).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                    Common Packet Header                       /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Coded Symbol ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                       Encoding Vector                         /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Encoded Payload Size      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
/                          Payload                              /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
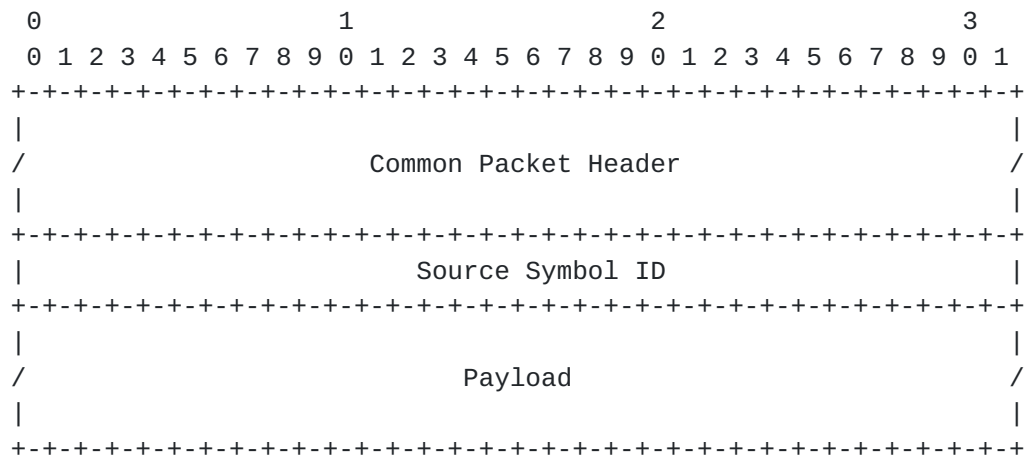
Figure 5: Coded Packet Format

Common Packet Header: a common packet header (as common header
format) where Packet Type=1.

Coded Symbol ID: the sequence number to identify a coded symbol.

Encoding Vector: an encoding vector to define the linear combination
used (coefficients, and source symbols).

Encoded Payload Size: the coded payload size used if the source
symbols have variable size (optional, Section 6.1.1.2)).

Payload: the coded symbol.

4.4. Acknowledgement Packet Format

A Tetrys Decoding Building Block or Tetrys Recoding Building Block
MAY send back to another building block some Acknowledgement packets.
They contain information about what it has received and/or decoded,
and other information such as a packet loss rate or the size of the
decoding buffers. The acknowledgement packets are OPTIONAL hence
they could be omitted or lost in transmission without impacting the
protocol behavior.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                     Common Packet Header                      /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Nb of missing source symbols                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Nb of not already used coded symbols            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     First Source Symbol ID                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   SACK size   |                                               |
+-+-+-+-+-+-+-+-+-+                                             +
|                                                               |
/                        SACK Vector                            /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Acknowledgement Packet Format

Common Packet Header: a common packet header (as common header
format) where Packet Type=2.

Nb missing source symbols: the number of missing source symbols in
the receiver since the beginning of the session.

Nb of not already used coded symbols: the number coded symbols at the
receiver that have not already been used for decoding (e.g., the
linear combinations contains at least 2 unknown source symbols).

First Source Symbol ID: ID of the first source symbol to consider for
acknowledgement.

SACK size: the size of the SACK vector in 32-bit words. For
instance, with value 2, the SACK vector is 64 bits long.

SACK vector: bit vector indicating the acknowledged symbols from the
first source symbol ID. The "First Source Symbol" is included in
this bit vector. A bit equal to 1 at position i means that the
source symbol of ID equal to "First Source Symbol ID" + i is
acknowledged by this acknowledgment packet.

5. The Coding Coefficient Generator Identifiers

5.1. Definition

The Coding Coefficient Generator Identifiers define a function or an
algorithm to build the coding coefficients used to generate the coded
symbols. They MUST be known by all the Tetrys encoders, recoders or
decoders.

5.2. Table of Identifiers

0000: GF256 (or GF(2**8)) Vandermonde based coefficients. Each
coefficient is built as alpha^((source_symbol_id*coded_symbol_id) %
255).

0001: GF16 (or GF(2**4)) Vandermonde based coefficients. Each
coefficient is built as alpha^((source_symbol_id*coded-symbol_id) %
15).

0010: SRLC.

Others: To be discussed.

6. Tetrys Basic Functions

6.1. Encoding

At the beginning of a transmission, a Tetrys Encoding Building Block
or MUST choose an initial code rate (added redundancy) as it doesn't
know the packet loss rate of the channel. In steady state, depending
on the code-rate, the Tetrys Encoding Building Block CAN generate
coded symbols when it receives a source symbol from the application
or some feedback from the decoding or recoding blocks.

When a Tetrys Encoding Building Block needs to generate a coded
symbol, it considers the set of source symbols stored in the Elastic
Encoding Window. These source symbols are the set of source symbols
which are not yet acknowledged by the receiver.

A Tetrys Encoding Building Block SHOULD set a limit to the Elastic
Encoding Window maximum size. This allows to control the algorithmic
complexity at the encoder and decoder by limiting the size of linear
combinations. It is also needed in situations where acknowledgment
packets are all lost or absent.

At the generation of a coded symbol, the Tetrys Encoding Building
Block generates an encoding vector containing the IDs of the source
symbols stored in the Elastic Encoding Window. For each source

symbol, a finite field coefficient is determined using a Coding
Coefficient Generator. This generator CAN take as input the source
symbol ID and the coded symbol ID and CAN determine a coefficient in
a deterministic way. A classical example of such deterministic
function is a generator matrix where the rows are indexed by the
source symbol IDs and the columns by the coded symbol IDs. For
example, the entries of this matrix can be built from a Vandermonde
structure, like Reed-Solomon codes, or from a sparse binary matrix,
like Low-Density Generator Matrix codes. Finally, the coded symbol
is the sum of the source symbols multiplied by their corresponding
coefficients.

6.1.1. Encoding Vector Formats

Each coded packet contains an encoding vector. The encoding vectors
CAN contain the ID and/or coefficient of each source symbol contained
in the coded symbol.

6.1.1.1. Transmitting the source symbol IDs

The source symbol IDs are organized as a sorted list of 32-bit
unsigned integers. Depending on the feedback, the source symbol IDs
can be successive or not in the list.

If they are successive, only the boundaries CAN be stored in the
encoding vector: it just needs 2*32-bit of information.

If not, the edge blocks CAN be stored directly, or a differential
transform to reduce the number of bits needed to represent an ID CAN
be used.

6.1.1.1.1. Compressed list of Source symbol IDs

Assume the symbol IDs used in the combination are:
[1..3],[5..6],[8..10].

1. Keep the first element in the packet as the first_source_id: 1.

2. Apply a differential transform to the others elements
    ([3,5,6,8,10]) which removes the element i-1 to the element i,
    starting with the first_source_id as i0, and get the list L =>
    [2,2,1,2,2]

3. Compute b, the number of bits needed to store all the elements,
    which is ceil(log2(max(L))): here, 2 bits.

4. Write b in the corresponding field, and write all the b * [(Nb
    IDs) - 1] elements in a bit vector, here: 10 10 01 10 10.

6.1.1.1.2. **Decompressing the Source symbol IDs**

   When a Tetrys Decoding Building Block wants to reverse the
   operations, this algorithm is used:

   1. Rebuild the list of the transmitted elements by reading the bit
       vector and b: [10 10 01 10 10] => [2,2,1,2,2]

   2. Apply the reverse transform by adding successively the elements,
       starting with first_source_id: [1,1+2,(1+2)+2,(1+2+2)+1,...] =>
       [1,3,5,6,8,10]

   3. Rebuild the IDs using the list and first_source_id:
       [1..3],[5..6],[8..10].

6.1.1.2. **Encoding Vector Format**

   The encoding vector CAN be used to store the source symbol IDs
   included in the associated coded symbol, the coefficients used in the
   combination, or both. It CAN be used to send only the number of
   source symbols included in the coded symbol.

   The encoding vector format uses a 4-bit Coding Coefficient Generator
   Identifier to identity the algorithm to generate the coefficients,
   and contains a set of IDs from the source symbol used in the
   combination. In this format, the number of IDs is stored as a 8-bit
   unsigned integer. To reduce the overhead, a compressed way to store
   the symbol IDs CAN be used: the IDs are not stored as themselves, but
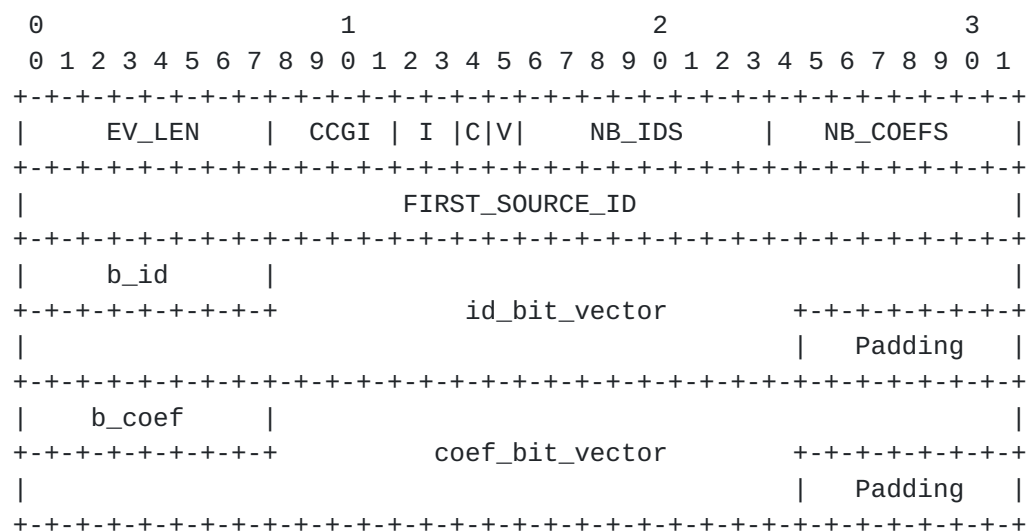   stored as the difference between the previous.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     EV_LEN    |  CCGI | I |C|V|    NB_IDS     |    NB_COEFS    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       FIRST_SOURCE_ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     b_id     |                                                |
+-+-+-+-+-+-+-+-+          id_bit_vector      +-+-+-+-+-+-+-+-+
|                                             |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     b_coef    |                                              |
+-+-+-+-+-+-+-+-+          coef_bit_vector     +-+-+-+-+-+-+-+-+
|                                             |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                   Figure 7: Encoding Vector Format

o Encoding Vector Length (EV_LEN) (8-bits): size in units of 32-bit
   words.

o Coding Coefficient Generator Identifier (CCGI): 4-bit ID to
   identify the algorithm or the function used to generate the
   coefficients (see Section 5). As a CCGI is included in each
   encoded vector, it can dynamically change between the generation
   of 2 coded symbols.

o Store the Source symbol IDs (I) (2 bits):

   * 00 means there is no source symbol ID information.

   * 01 means the encoding vector contains the edge blocks of the
      source symbol IDs without compression.

   * 10 means the encoding vector contains the compressed list of
      the source symbol IDs.

   * 11 means the encoding vector contains the compressed edge
      blocks of the source symbol IDs.

o Store the coefficients (C): 1 bit to know if an encoding vector
   contains information about the coefficients used.

o Having source symbols with variable size (V): set V to 1 if the
   combination which refers the encoding vector is a combination of
   source symbols with variable sizes. In this case, the coded
   packets MUST have the 'Encoded Payload Size' field.

o Number of IDs used to store the source symbol IDs (NB_IDS): the
   number of IDs stored in the encoding vector (depending on I).

o Number of coefficients (NB_COEFS): The number of the coefficients
   used to generate the associated coded symbol.

o The first source Identifier (FIRST_SOURCE_ID): the first source
   symbol ID used in the combination.

o Number of bits for each ID (b_id): the number of bits needed to
   store the IDs (see Section 6.1.1.1).

o Information about the source symbol IDs (id_bit_vector): if I=01,
   store the edge blocks as b_id * (NB_IDS - 1). If I=10, store in a
   compressed way the edge blocks.

o Number of bits needed to store each coefficient (b_coef): the
   number of bits used to store the coefficients.

   o The coefficients (coef_bit_vector): The coefficients stored (as a
      vector of b_coef * NB_COEFS).

   o Padding: padding to have an Encoding Vector size multiple of
      32-bit (for the id and coefficient part).

6.2. **The Elastic Encoding Window**

   When an input source symbol is passed to a Tetrys Encoding Building
   Block, it is added to the Elastic Encoding Window. This window MUST
   have a limit set by the encoding building Block (depending of the use
   case: unicast, multicast, file transfer, real-time transfer, ...).
   If the Elastic Encoding Window reached its limit, the window slides
   over the symbols: the first (oldest) symbols are removed. Then, a
   packet containing this symbol can be sent onto the network. As an
   element of the coding window, this symbol is included in the next
   linear combinations created to generate the coded symbols.

   As explained below, the receiver or the recoder sends periodic
   feedback indicating the received or decoded source symbols. In the
   case of a unicast transmission, when the sender receives the
   information that a source symbol was received and/or decoded by the
   receiver, it removes this symbol from the coding window.

   In a multicast transmission:

   o If the acknowledgement packets are not enabled, the coding window
      grows up to a limit. When the limit is reached, the oldest
      symbols are removed from the coding window.

   o If the acknowledgement packets are enabled, a source symbol is
      removed from the coding window when all the receivers have
      received or decoded it or when the coding window reaches its
      limit.

6.3. **Recoding**

6.3.1. **Principle**

   A Tetrys Recoding Block maintains a list of the ID of the source
   symbols included in the Elastic Coding Window of the sender. It also
   stores a set of received source and coded symbols able to regenerate
   the set or a subset of the symbols of the Elastic Coding Window. In
   other words, if R1, ..., Rt represent t received symbols and S1, ...,
   Sk represent the set or a subset of the source symbols of the Elastic
   Coding Window, there exists a t*k-matrix M such that (R1, ..., Rt).M
   = (S1, ..., Sk).

6.3.2. Generating a coded symbol at an intermediate node

At the generation of a coded symbol, the Tetrys Recoding Building
Block generates an encoding vector containing the IDs of the source
symbols stored in the Elastic Encoding Window or in the subset of the
Elastic Encoding Window that it is able to regenerate. The Tetrys
Recoding Building Block then generates a new coded symbol ID
different from the received coded symbol IDs. From this coded symbol
ID and the source symbol IDs of (S1, ..., Sk), a finite field
coefficient is determined using a Coding Coefficient Generator. Let
(a1, ...,ak) denote the obtained coefficients. To compute the linear
combination (s1, ..., Sk).transpose(a1, ..., ak) the Tetrys Recoding
Building block computes the vector v = (a1, ...,ak).transpose(M) and
then computes the coded symbol R = (R1, ..., Rt).transpose(v). It
can be verified that the new coded symbol is obtained without any
decoding operation.

6.4. Decoding

A classical matrix inversion is sufficient to recover the source
symbols.

7. Security Considerations

N/A

8. Privacy Considerations

N/A

9. IANA Considerations

N/A

10. Acknowledgments

N/A

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119,
           DOI 10.17487/RFC2119, March 1997,
           <https://www.rfc-editor.org/info/rfc2119>.

[11.2](#). Informative References

[AHL-00]   Ahlswede, R., Ning Cai, Li, S., and R. Yeung, "Network
            information flow", IEEE Transactions on Information
            Theory vol.46, no.4, pp.1204,1216, July 2000.

[FECFRAME]
            Watson, M., Begen, A., and V. Roca, "Forward Error
            Correction (FEC) Framework", Request for Comments 6363,
            October 2011.

[NWCRG-ARCH]
            NWCRG, "Network Coding Architecture", TBD TBD.

[RMT]   Vicisano, L., Gemmel, J., Rizzo, L., Handley, M., and J.
            Crowcroft, "Forward Error Correction (FEC) Building
            Block", Request for Comments 3452, December 2002.

[Tetrys]   Lacan, J. and E. Lochin, "Rethinking reliability for long-
            delay networks", International Workshop on Satellite and
            Space Communications 2008 (IWSSC08), October 2008.

Authors' Addresses

    Jonathan Detchart
    ISAE
    10, avenue Edouard-Belin
    BP 54032
    Toulouse CEDEX 4 31055
    France

    Email: jonathan.detchart@isae.fr

    Emmanuel Lochin
    ISAE
    10, avenue Edouard-Belin
    BP 54032
    Toulouse CEDEX 4 31055
    France

    Email: emmanuel.lochin@isae.fr

Jerome Lacan
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4 31055
France

Email: jerome.lacan@isae.fr

Vincent Roca
INRIA
655, av. de l'Europe
Inovallee; Montbonnot
ST ISMIER cedex 38334
France

Email: vincent.roca@inria.fr