Workgroup: NWCRG
Internet-Draft: draft-irtf-nwcrg-tetrys-03
Published: 6 September 2022
Intended Status: Experimental
Expires: 10 March 2023
Authors: J. Detchart (ISAE-SUPAERO), E. Lochin (ENAC), J. Lacan (ISAE-SUPAERO), V. Roca (INRIA)

Tetrys, an On-the-Fly Network Coding Protocol

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 may recover from erasures within an RTT-independent delay, thanks to the transmission of Coded Packets. This document is a record of the experience gained by the authors while developing and testing in real conditions the Tetrys protocol.

This document is a product of the Coding for Efficient Network Communications Research Group (NWCRG). It conforms to the NWCRG taxonomy[RFC8406].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document is a product of and represents the collaborative work and consensus of the Coding for Efficient Network Communications Research Group (NWCRG). It is not an IETF product and is not an IETF standard.

This document describes Tetrys, a novel erasure 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 [RFC3452] and the FECFRAME [RFC8680] working groups. The protocol presented here may be seen as a network coding extension to standard unicast transport protocols (or even multicast or anycast with a few modifications). The current proposal may be considered a combination of network erasure coding and feedback mechanisms [Tetrys], [Tetrys-RT] .

The main innovation of the Tetrys protocol is in the generation of Coded Packets from an Elastic Encoding Window. This window is filled by any Source Packets coming from an input flow and is periodically updated with the receiver's feedbacks. These feedbacks return to the sender the highest sequence number received or rebuilt, which allows to flush the corresponding Source Packets stored in the encoding window. The size of this window may be fixed or dynamically updated. If the window is full, incoming Source Packets replace older sources packets which are dropped. As a matter of fact, its limit should be correctly sized. Finally, Tetrys allows to deal with losses on both the forward and return paths and in particular, is resilient to acknowledgment losses. All these operations are further detailed in Section Section 4.

With Tetrys, a Coded Packet is a linear combination over a finite field of the data Source Packets belonging to the coding window. The coefficients finite field's choice is a trade-off between the best erasure recovery performance (finite fields of 256 elements) and the system constraints (finite fields of 16 elements is preferred) and is driven by the application.

Thanks to the Elastic Encoding Window, the Coded Packets are built on-the-fly, by using a predefined method to choose the coefficients. The redundancy ratio may be dynamically adjusted, and the coefficients may be generated in different ways, during the transmission. Compared to FEC block codes, this allows reducing the bandwidth use and the decoding delay.

The design of Tetrys protocol detailed in this document is completed by a record of the experience gained by the authors while developing and testing in real conditions the Tetrys protocol. In particular, several research issues are discussed in Section 6 following our own experience and observations.

1.1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Definitions, Notations and Abbreviations

The notation used in this document is based on the NWCRG taxonomy [RFC8406] .

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 Coded Symbols.

Encoding Vector: a set of the coding 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 Encoder.

Output Symbol: a symbol generated by the Tetrys Encoder. For a non-systematic mode, all Output Symbols are Coded Symbols. For a systematic mode, Output Symbols MAY 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 MAY 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 involved in the coding process.

Coding Coefficient Generator Identifier: a unique identifier that defines 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

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 at a single encoding node. Note that the input stream MAY be a multiplex of several upper layer streams. Transmission MAY be over a single path or multiple paths. This is the simplest use-case, that is very much aligned with currently proposed scenarios for end-to-end streaming.

3.2. Overview

```
+----------+                +----------+
|          |                |          |
|    App   |                |    App   |
|          |                |          |
+----------+                +----------+
     |                           ^
     |  Source           Source  |
     |  Symbols          Symbols |
     |                           |
     v                           |
+----------+                +----------+
|          | output packets |          |
|  Tetrys  |--------------->|  Tetrys  |
|  Encoder |Feedback Packets|  Decoder |
|          |<---------------|          |
+----------+                +----------+
```

Figure 1: Tetrys Architecture

The Tetrys protocol features several key functionalities. The mandatory features are:

*on-the-fly encoding;

*decoding;

*signaling, to carry in particular the symbol identifiers in the encoding window and the associated coding coefficients when meaningful;

*feedback management;

*elastic window management;

*Tetrys packet header creation and processing;

and the optional features are :

*channel estimation;

*dynamic adjustment of the Code Rate and flow control;

*congestion control management (if appropriate). See Section 6.1 for further details;

Several building blocks provide these functionalities:

*The Tetrys Building Block: this BB embeds both the Tetrys Decoder and Tetrys Encoder and thus, is used during encoding, 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 may be used.

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

To ease the addition of future components and services, Tetrys adds a header extension mechanism, compatible with that of LCT [RFC5651], NORM [RFC5740], FECFRAME [RFC8680].

4. Tetrys Basic Functions

4.1. Encoding

At the beginning of a transmission, a Tetrys Encoder MUST choose an initial Code Rate (added redundancy) as it doesn't know the packet loss rate of the channel. In the steady state, depending on the Code Rate, the Tetrys Encoder MAY generate Coded Symbols when it receives a Source Symbol from the application or some feedback from the decoding blocks.

When a Tetrys Encoder needs to generate a Coded Symbol, it considers the set of Source Symbols stored in the Elastic Encoding Window and generates an Encoding Vector with the Coded Symbol. These Source Symbols are the set of Source Symbols that are not yet acknowledged by the receiver. For each Source Symbol, a finite field coefficient is determined using a Coding Coefficient Generator. This generator MAY take as input the Source Symbol IDs and the Coded Symbol ID and MAY determine a coefficient in a deterministic way as presented in Section 5.3. Finally, the Coded Symbol is the sum of the Source Symbols multiplied by their corresponding coefficients.

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

4.2. The Elastic Encoding Window

When an input Source Symbol is passed to a Tetrys Encoder, it is added to the Elastic Encoding Window. This window MUST have a limit set by the encoding building Block. If the Elastic Encoding Window reached its limit, the window slides over the symbols: the first (oldest) symbol is removed, and the newest symbol is added. 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 Tetrys Decoder sends periodic feedback indicating the received or decoded Source Symbols. When the sender receives the information that a Source Symbol was received or decoded by the receiver, it removes this symbol from the coding window.

4.3. Decoding

A standard Gaussian elimination is sufficient to recover the erased Source Symbols, when the matrix rank enables it.

5. Packet Format

5.1. 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    |    PKT_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

As already noted above in the document, this format is inspired and inherits from the LCT header format [RFC5651] with slight modifications.

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

* 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.

* 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.

* 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.

* Header length (HDR_LEN): 8 bits. The 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 may be used to directly access the portion of the packet beyond the Tetrys header, i.e., to the first next 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 others headers or packet payload.

* PKT_TYPE: Tetrys packet type, 8 bits. Type of packet. There is 3 types of packets: the PKT_TYPE_SOURCE (0) defined in Section 5.2, the PKT_TYPE_CODED (1) defined in Section 5.3 and the PKT_TYPE_WND_UPT (3), for window update packets defined in Section 5.4.

* 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 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.

* Transport Session Identifier (TSI): 0 or 32 bits The TSI uniquely identifies a session among all sessions from a particular Tetrys encoder. 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, conjointly with the IP address of the sender, always uniquely identifies a session, whether 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.

5.1.1. 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 MAY 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

*Header Extension Type (HET): 8 bits The type of the Header Extension. This document defines several 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.

* 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).

* Header Extension Content (HEC): variable length The content of the Header Extension. The format of this subfield 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.

5.2. Source Packet Format

A Source Packet is a Common Packet Header encapsulation, a Source Symbol ID and a Source Symbol (payload). The Source Symbols MAY 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)

5.3. 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 MAY have variable sizes, all the Source Symbol sizes need to be encoded. To generate this encoded payload size, as a 16-bit unsigned value, the linear combination uses the same coefficients as the coded payload. 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 5.3.1).

```
 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 a variable size (optional,Section 5.3.1).

Payload: the Coded Symbol.

5.3.1. The Encoding Vector

An Encoding Vector contains all the information about the linear combination used to generate a Coded Symbol. The information includes the source identifiers and the coefficients used for each Source Symbol. It MAY be stored in different ways depending on the situation.

```
 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   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                          coef_bit_vector        +-+-+-+-+-+-+-+
|                                                 |   Padding   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Encoding Vector Format

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

* Coding Coefficient Generator Identifier (CCGI): 4-bit ID to identify the algorithm or the function used to generate the coefficients. As a CCGI is included in each encoded vector, it MAY dynamically change between the generation of 2 Coded Symbols. The CCGI builds the coding coefficients used to generate the Coded Symbols. They MUST be known by all the Tetrys encoders or decoders. The two RLC FEC schemes specified in this document reuse the Finite Fields defined in [RFC5510], Section 8.1. More specifically, the elements of the field GF(2^(m)) are represented by polynomials with binary coefficients (i.e., over GF(2)) and degree lower or equal to m-1. The addition between two elements is defined as the addition of binary polynomials in GF(2), which is equivalent to a bitwise XOR operation on the binary representation of these elements. With GF(2^(8)), multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial is used for GF(2^(8)): x^(8) + x^(4) + x^(3) + x^(2) + 1 With GF(2^(4)), multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 4.

The following irreducible polynomial is used for GF(2^(4)): x^(4) + x + 1

- -0: Vandermonde based coefficients over the finite field GF(2^(4)), as defined below. Each coefficient is built as alpha^((source_symbol_id*coded-symbol_id) % 16), with alpha the root of the primitive polynomial.

- -1: Vandermonde based coefficients over the finite field GF(2^(8)), as defined below. Each coefficient is built as alpha^((source_symbol_id*coded-symbol_id) % 256), with alpha the root of the primitive polynomial.

- -Suppose we want to generate the Coded Symbol 2 as a linear combination of the Source Symbols 1,2,4 using CCGI=1. The coefficients will be alpha^((1 * 1) % 256), alpha^((1 * 2) % 256), alpha^((1 * 4) % 256).

*Store the Source Symbol ID Format (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.

*Store the Encoding Coefficients (C): 1 bit to indicate if an Encoding Vector contains information about the coefficients used.

*Having Source Symbols with Variable Size Encoding (V): set V to 1 if the combination which refers to 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.

*NB_IDS: the number of source IDs stored in the Encoding Vector (depending on I).

*Number of coefficients (NB_COEFS): The number of the coefficients used to generate the associated Coded Symbol.

*The first source identifier (FIRST_SOURCE_ID): the first Source Symbol ID used in the combination.

*Number of bits for each edge block (b_id): the number of bits needed to store the edge.

* Information about the Source Symbol IDs (id_bit_vector): if I=01, store the edge blocks as b_id * (NB_IDS * 2 - 1). If I=10, store in a compressed way the edge blocks.

* The coefficients (coef_bit_vector): The coefficients stored depending on the CCGI (4 or 8 bits for each coefficient).

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

The Source Symbol IDs are organized as a sorted list of 32-bit unsigned integers. Depending on the feedback, the Source Symbol IDs MAY be successive or not in the list. If they are successive, the boundaries are stored in the Encoding Vector: it just needs 2*32-bit of information. If not, the full list or the edge blocks MAY be stored, and a differential transform to reduce the number of bits needed to represent an identifier MAY be used.

For the following subsections, let's take as an example the generation of an encoding vector for a Coded Symbol which is a linear combination of the Source Symbols with IDs 1,2,3,5,6,8,9 and 10 (or as edge blocks: [1..3],[5..6],[8..10])

There are several ways to store the Source Symbols IDs into the encoding vector:

* If no information about the Source Symbol IDs is needed, the field I MUST be set to 0b00: no b_id and no id_bit_vector field

* If the edge blocks are stored without compression, the field I MUST be set to 0b01. In this case, set b_id to 32 (as a symbol id is 32 bits), and store into id_bit_vectors the list as 32 bits unsigned integers: 1,3,5,6,8,10

* If the Source Symbols Ids are stored as a list with compression, the field I MUST be set to 0b10. In this case, see the section Section 5.3.1.1 but rather than compressing the edge blocks, we compress the full list of the Source Symbol IDs.

* If the edge blocks are stored with compression, the field I MUST be set to 0b11. In this case, see the section Section 5.3.1.1.

5.3.1.1. Compressed list of Source Symbol IDs

Let's continue with our Coded Symbol defined in the previous section. The Source Symbols IDs used in the linear combination are: [1..3],[5..6],[8..10].

If we want to compress and store this list into the encoding vector, we MUST follow this procedure:

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

2. Apply a differential transform to the other 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))), where max(L) represents the maximum of the elements of the list L: here, 2 bits.

4. Write b in the corresponding field, and write all the b * [(2 * NB blocks) - 1] elements in a bit vector, here: 10 10 01 10 10.

5.3.1.2. Decompressing the Source Symbol IDs

When a Tetrys Decoding 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 blocks using the list and first_source_id: [1..3], [5..6],[8..10].

5.4. Window Update Packet Format

A Tetrys Decoder MAY send back to another building block some Window Update packets. They contain information about what the packets received, decoded or dropped, and other information such as a packet loss rate or the size of the decoding buffers. They are used to optimize the content of the encoding window. The window update packets are OPTIONAL, and 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_missing_src                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   nb_not_used_coded_symb                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         first_src_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      plr      |   sack_size   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
/                         SACK Vector                           /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Window Update Packet Format

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

nb_missing_src: the number of missing Source Symbols in the receiver since the beginning of the session.

nb_not_used_coded_symb: the number of Coded Symbols at the receiver that have not already been used for decoding (e.g., the linear combinations contain at least 2 unknown Source Symbols).

first_src_id: ID of the first Source Symbol to consider in the SACK vector.

plr: packet loss ratio expressed as a percentage normalized to a 8-bit unsigned integer. For example, 2.5 % will be stored as floor(2.5 * 256/100) = 6. Conversely, if 6 is the stored value, the corresponding packet loss ratio expressed as a percentage is 6*100/256 = 2.34 %. This value is used in the case of dynamic Code Rate or for statistical purpose. The choice of calculation is left to the Tetrys Decoder, depending on a window observation, but should be the PLR seen before decoding.

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 symbols that must be removed in the encoding window from the first Source Symbol ID. In most cases, these symbols were received by the receiver. The other cases concern some events with non-recoverable packets (for example in the case of a burst of losses) where it is better to drop and abandon some packets, and thus to remove them from the encoding window, to allow the recovery of the following packets. The "First Source Symbol" is included in this bit vector. A bit equal to 1 at the i-th position means that this window update packet removes the Source Symbol of ID equal to "First Source Symbol ID" + i from the encoding window.

6. Research Issues

The present document describes the baseline protocol, allowing communications between a Tetrys encoder and a Tetrys decoder. In practice, Tetrys can be used either as a standalone protocol or embedded inside an existing protocol, and either above, within or below the transport layer. All these situations raise manyfold research questions to come up with a complete protocol solution, that we briefly discuss hereafter.

6.1. Interaction with Congestion Control

The Tetrys and congestion control components generate two separate channels (see [I-D.irtf-nwcrg-coding-and-congestion], section 2.1):

* the Tetrys channel carries source and Coded Packets (from the sender to the receiver) and information from the receiver to the sender (e.g., signaling which symbols have been recovered, loss rate prior and/or after decoding, etc.);

* the congestion control channel carries packets from a sender to a receiver, and packets signaling information about the network (e.g., number of packets received versus lost, Explicit Congestion Notification (ECN) marks, etc.) from the receiver to the sender.

In practice, depending on how Tetrys is deployed (i.e., above, within or below the transport layer), [I-D.irtf-nwcrg-coding-and-congestion] identifies and discusses several topics. They are briefly listed below and adapted to the particular case of Tetrys:

* congestion related losses may be hidden if Tetrys is deployed below the transport layer without any precaution (i.e., Tetrys recovering packets lost because of a congested router), which can severely impact the the congestion control efficiency. An approach is suggested to avoid hiding such signals in [I-D.irtf-nwcrg-coding-and-congestion], section 5;

* having Tetrys and non-Tetrys flows sharing the same network links can raise fairness issues between these flows. The situation depends in particular on whether some of these flows are congestion controlled and not others, and which type of

congestion control is used. The details are out of scope of this document, but may have major impacts in practice;

* coding rate adaptation within Tetrys can have major impacts on congestion control if done inappropriately. This topic is discussed more in detail in Section 6.2;

* Tetrys can leverage on multipath transmissions, the Tetrys packets being sent to the same receiver through multiple paths. Since paths can largely differ, a per-path flow control and congestion control adaptation could be needed;

* protecting several application flows within a single Tetrys flow raises additional questions. This topic is discussed more in detail in Section 6.3.

6.2. Adaptive Coding Rate

When the network conditions (e.g., delay and loss rate) strongly vary over time, an adaptive coding rate can be used to increase or reduce the amount of Coded Packets among a transmission dynamically (i.e., the added redundancy), with the help of a dedicated algorithm, similarly to [A-FEC]. Once again, the strategy differs, depending on which layer Tetrys is deployed (i.e., above, within or below the transport layer). Basically, we can slice these strategies in two distinct classes: when Tetrys is deployed inside the transport layer, versus outside (i.e., above or below). A deployment within the transport layer obviously means that interactions between transport protocol micro-mechanisms, such as the error recovery mechanism, the congestion control, the flow control or both, are envisioned. Otherwise, deploying Tetrys within a non congestion controlled transport protocol, like UDP, would not bring out any other advantage than deploying it below or above the transport layer.

The impact deploying a FEC mechanism within the transport layer is further discussed in [I-D.irtf-nwcrg-coding-and-congestion], section 4, where considerations concerning the interactions between congestion control and coding rates, or the impact of fairness, are investigated. This adaptation may be done jointly with the congestion control mechanism of a transport layer protocol, as proposed by [CTCP]. This allows the use of monitored congestion control metrics (e.g., RTT, congestion events, or current congestion window size) to adapt the coding rate conjointly with the computed transport sending rate. The rationale is to compute an amount of repair traffic that does not lead to congestion. This joint optimization is mandatory to prevent flows to consume the whole available capacity as also discussed in [I-D.singh-rmcat-adaptive-fec] where the authors point out that an increase in the repair ratio should be done conjointly with a decrease in the source sending rate.

Finally, adapting a coding rate can also be done outside the transport layer and without considering transport layer metrics. In particular, this adaptation may be done jointly with the network as proposed in [RED-FEC]. In this paper, the authors propose a Random Early Detection FEC mechanism in the context of video transmission over wireless networks. Briefly, the idea is to add more redundancy packets if the queue at the access point is less occupied and vice versa. A first theoretical attempt for video delivery has been proposed [THAI] with Tetrys. This approach is interesting as it illustrates a joint collaboration between the application requirements and the network conditions and combines both signals coming from the application needs and the network state (i.e., signals below or above the transport layer).

To conclude, there are multiple ways to enable an adaptive coding rate. However, all of them depend on:

* the signal metrics that can be monitored and used to adapt the coding rate;
* the transport layer used, whether congestion controlled or not;
* the objective sought (e.g., to minimize congestion, or to fit application requirements).

6.3. Using Tetrys Below The IP Layer For Tunneling

The use of Tetrys to protect an aggregate of flows, typically when Tetrys is used for tunneling, to recover from IP datagram losses, raises research questions. When redundancy is applied without flow differentiation, this may come in contradiction with the service requirements of individual flows, some of them may be more penalized by high latency and jitter than by partial reliability, while other flows may have opposite requirements. In practice head-of-line blocking will impact all flows in a similar manner despite their different needs, which asks for more elaborate strategies inside Tetrys.

7. Security Considerations

Tetrys inherits a subset of the security issues described in FECFRAME [RFC8680] and in particular in sections "9.2.2. Content Corruption" and "9.3. Attacks against the FEC Parameters". As an application layer end-to-end protocol, security considerations of Tetrys should also be comparable to those of HTTP/2 with TLS. The considerations from Section 10 of HTTP2 [RFC7540] also apply in addition to those listed here.

8. IANA Considerations

This document does not ask for any IANA registration.

9. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 7942 before publishing the RFC. Thanks!

An implementation of Tetrys exists:

organization: ISAE-SUPAERO

Description: This is a proprietary implementation made by ISAE-SUPAERO

Maturity: "production"

Coverage: this software implements TETRYS with some modifications

Licensing: proprietary

Implementation experience: maximum

Information update date: January 2022

Contact: jonathan.detchart@isae-supaero.fr

10. Acknowledgments

First, the authors want sincerely to thank Marie-Jose Montpetit for continuous help and support on Tetrys. Marie-Jo, many thanks!

The authors also wish to thank NWCRG group members for numerous discussions on on-the-fly coding that helped finalize this document.

Finally, the authors would like to thank Colin Perkins for providing comments and feedback on the document.

11. References

11.1. Normative References

[I-D.irtf-nwcrg-coding-and-congestion] Kuhn, N., Lochin, E., Michel, F., and M. Welzl, "Coding and congestion control in transport", Work in Progress, Internet-Draft, draft-irtf-nwcrg-coding-and-congestion-12, 22 February 2022,

<https://www.ietf.org/archive/id/draft-irtf-nwcrg-coding-and-congestion-12.txt>.

[RFC2119] Bradner, S., "Keywords 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>.

[RFC3452] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "Forward Error Correction (FEC) Building Block", RFC 3452, DOI 10.17487/RFC3452, December 2002, <https://www.rfc-editor.org/info/rfc3452>.

[RFC5510] Lacan, J., Roca, V., Peltotalo, J., and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes", RFC 5510, DOI 10.17487/RFC5510, April 2009, <https://www.rfc-editor.org/info/rfc5510>.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, DOI 10.17487/RFC5651, October 2009, <https://www.rfc-editor.org/info/rfc5651>.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, DOI 10.17487/RFC5740, November 2009, <https://www.rfc-editor.org/info/rfc5740>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8406]

Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Ghanem, S., Lochin, E., Masucci, A., Montpetit, M-J., Pedersen, M., Peralta, G., Roca, V., Ed., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, DOI 10.17487/RFC8406, June 2018, <https://www.rfc-editor.org/info/rfc8406>.

[RFC8680] Roca, V. and A. Begen, "Forward Error Correction (FEC) Framework Extension to Sliding Window Codes", RFC 8680, DOI 10.17487/RFC8680, January 2020, <https://www.rfc-editor.org/info/rfc8680>.

11.2. Informative References

[A-FEC] Bolot, J., Fosse-Parisis, S., and D. Towsley, "Adaptive FEC-based error control for Internet telephony", IEEE INFOCOM 99, pp. 1453-1460 vol. 3 DOI 10.1109/INFCOM. 1999.752166, 1999.

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

[CTCP] Kim (et al.), M., "Network Coded TCP (CTCP)", arXiv 1212.2291v3, 2013.

[I-D.singh-rmcat-adaptive-fec] Singh, V., Nagy, M., Ott, J., and L. Eggert, "Congestion Control Using FEC for Conversational Media", Work in Progress, Internet-Draft, draft-singh-rmcat-adaptive-fec-03, 20 March 2016, <https://www.ietf.org/archive/id/draft-singh-rmcat-adaptive-fec-03.txt>.

[RED-FEC] Lin, C., Shieh, C., Chilamkurti, N. K., Ke, C., and H. S. Hwang, "A RED-FEC Mechanism for Video Transmission Over WLANs", IEEE Transactions on Broadcasting, vol. 54, no. 3, pp. 517-524 DOI 10.1109/TBC.2008.2001713, September 2008.

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

[Tetrys-RT] Tournoux, P.U., Lochin, E., Lacan, J., Bouabdallah, A., and V. Roca, "On-the-fly erasure coding for real-time video applications", IEEE Transactions on Multimedia, Vol 13, Issue 4, August 2011 (TMM.2011), August 2011.

[THAI] Tran-Thai, T., Lacan, J., and E. Lochin, "Joint on-the-fly network coding/video quality adaptation for real-time delivery", Signal Processing: Image Communication, vol. 29 (no. 4), pp. 449-461 ISSN 0923-5965, 2014.

Authors' Addresses

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

Email: jonathan.detchart@isae-supaero.fr

Emmanuel Lochin
ENAC
7, avenue Edouard Belin
31400 Toulouse
France

Email: emmanuel.lochin@enac.fr

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

Email: jerome.lacan@isae-supaero.fr

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

Email: vincent.roca@inria.fr