Internet Research Task Force (IRTF)
Request for Comments: 9407
Category: Experimental
ISSN: 2070-1721

J. Detchart
ISAE-SUPAERO
E. Lochin
ENAC
J. Lacan
ISAE-SUPAERO
V. Roca
INRIA

June 2023

Tetrys: An On-the-Fly Network Coding Protocol

Abstract

This document describes Tetrys, which is an on-the-fly network coding protocol that can be used to transport delay-sensitive 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 the Tetrys protocol in real conditions.

This document is a product of the Coding for Efficient NetWork Communications Research Group (NWCRG). It conforms to the NWCRG taxonomy described in RFC 8406.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Coding for Efficient NetWork Communications Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9407.

carefully, as they describe your rights and restrictions with respect to this document.

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 or an IETF standard.

This document describes Tetrys, which is an on-the-fly network coding protocol that can be used to transport delay-sensitive and loss-sensitive data over a lossy network. 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 network codes have seen some deployment fairly recently in the Internet community, the use of application-layer erasure codes in the IETF has already been standardized in the RMT [RFC5052] [RFC5445] and 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 feedback. These feedback messages provide to the sender information about the highest sequence number received or rebuilt, which can enable the flushing 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 source packets that are dropped. As a matter of fact, its limit should be correctly sized. Finally, Tetrys allows dealing with losses on both the forward and return paths and is particularly resilient to acknowledgment losses. All these operations are further detailed in 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 choice of coefficients, as finite fields elements, is a trade-off between the best erasure recovery performance (finite fields of 256 elements) and the system constraints (finite fields of 16 elements are 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 Forward Error Correction (FEC) block codes, this reduces the bandwidth use and the decoding delay.

The design description of the Tetrys protocol in this document is complemented by a record of the experience gained by the authors while developing and testing the Tetrys protocol in realistic conditions. 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 BCP 14 [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 is 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 plus the encoding vectors.

Feedback Packet: A feedback packet is a packet containing information about the decoded or received source symbols. It MAY also contain 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 unacknowledged source packets of the input flow involved in the coding process.

Coding Coefficient Generator Identifier (CCGI): A unique identifier that defines a function or an algorithm allowing the generation of the encoding vector.

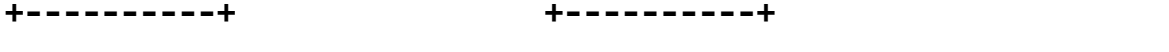
Code Rate: Defines the rate between the number of input symbols and the number of output symbols.

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 quite aligned with currently proposed scenarios for end-to-end streaming.

3.2. Overview



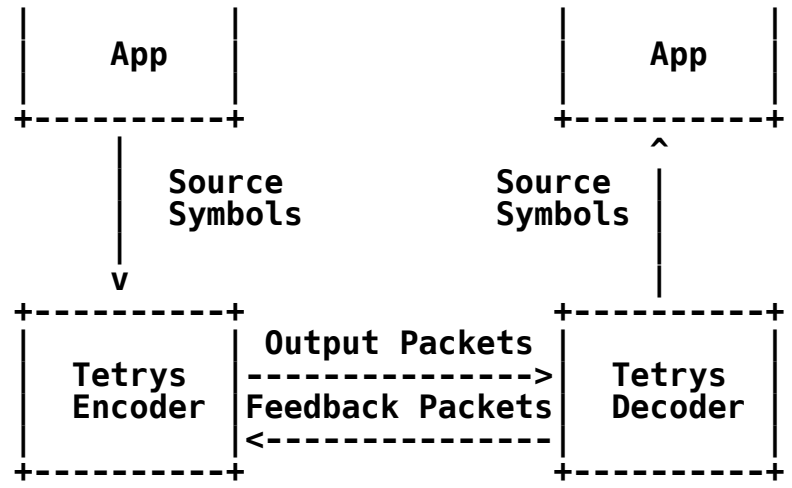


Figure 1: Tetrys Architecture

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

* on-the-fly encoding;

* decoding;

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

* feedback management;

* elastic window management; and

* Tetrys packet header creation and processing.

The optional features include:

* channel estimation;

* dynamic adjustment of the code rate and flow control; and

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

Several building blocks provide the following functionalities:

The Tetrys Building Block: This building block embeds both the Tetrys decoder and Tetrys encoder; thus, it 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 that is compatible with that of Layered Coding Transport (LCT) [RFC5651], NACK-Oriented Reliable Multicast (NORM) [RFC5740], and FEC Framework (FECFRAME) [RFC8680].

4. Tetrys Basic Functions

4.1. Encoding

At the beginning of a transmission, a Tetrys encoder MUST choose an initial code rate that adds redundancy as it doesn't know the packet loss rate of the channel. In the steady state, the Tetrys encoder MAY generate coded symbols when it receives a source symbol from the application or some feedback from the decoding blocks depending on the code rate.

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 the source symbol IDs and the coded symbol ID as an input 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 MUST 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 all window update packets are 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 has 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).

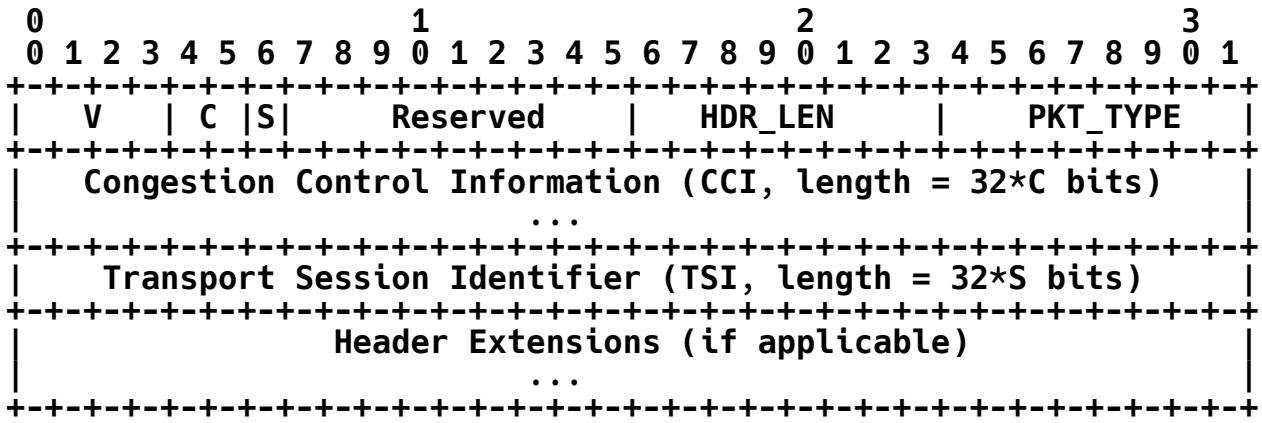


Figure 2: Common Header Format

As noted above, this format is inspired by, 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 set to 0b00 indicates the Congestion Control Information (CCI) field is 0 bits in length. C set to 0b01 indicates the CCI field is 32 bits in length. C set to 0b10 indicates the CCI field is 64 bits in length. C set to 0b11 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, 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.

Tetrys packet type (PKT_TYPE): 8 bits. There are three types of packets: the PKT_TYPE_SOURCE (0b00) defined in Section 5.2, the PKT_TYPE_CODED (0b01) defined in Section 5.3 and the PKT_TYPE_WND_UPT (0b11) 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 is set to 0b00. This field MUST be 32 bits if C is set to 0b01. This field MUST be 64 bits if C is set to 0b10. This field MUST be 96 bits if C is set to 0b11.

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; thus, the IP address of the sender and the TSI together uniquely identify the session. Although a TSI always uniquely identifies a session conjointly with the IP address of the sender, 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

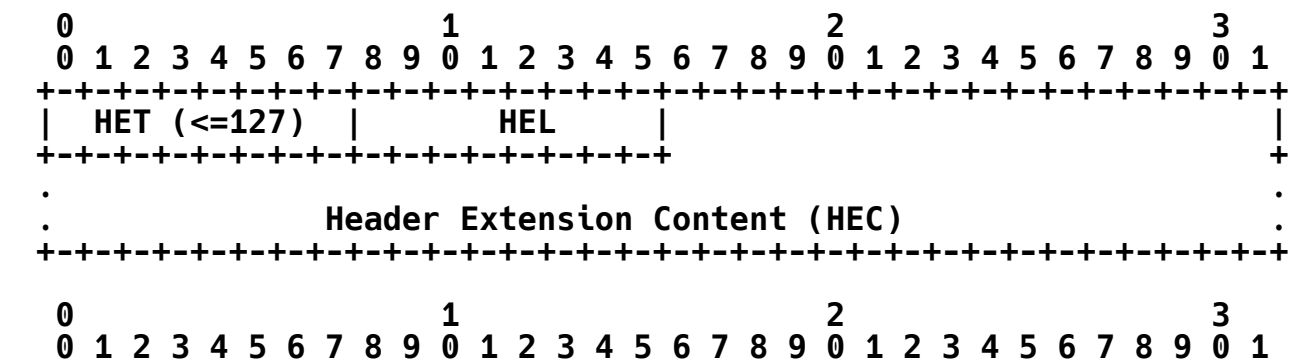
Header extensions are used in Tetrys to accommodate optional header fields that are not always used or have variable sizes. The presence of header extensions MAY be inferred by the Tetrys header length (HDR_LEN). If HDR_LEN is larger than the length of the standard header, then the remaining header space 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 for the future introduction of backward-compatible enhancements to Tetrys without changing the Tetrys version number. Header extensions that are not backward-compatible MUST NOT 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.



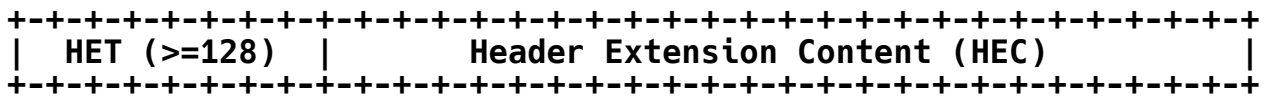


Figure 3: Header Extension Format

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): Length of the variable. 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 a variable size as specified by the HEL field. Note that the length of each header extension MUST be a multiple of 32 bits. Additionally, the total size of the Tetrys header, including all header extensions and 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.

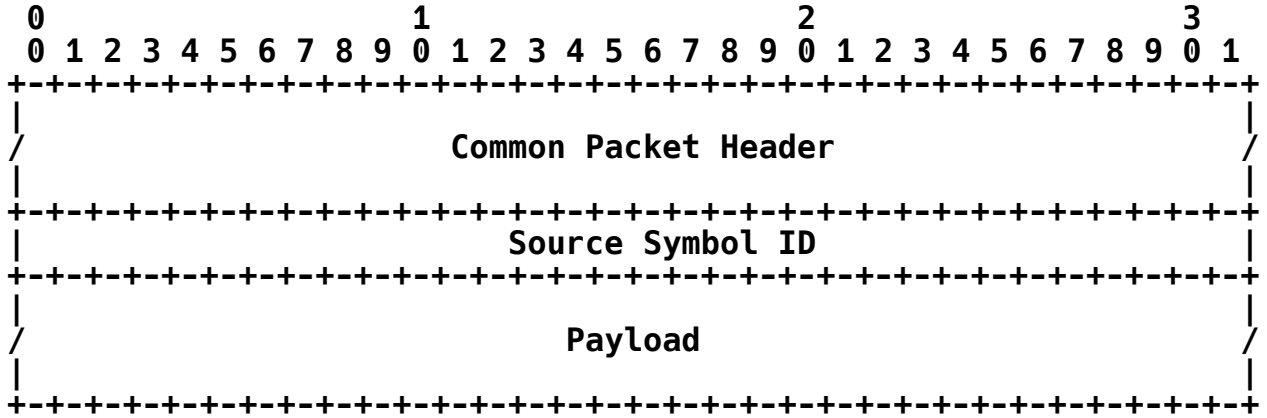


Figure 4: Source Packet Format

Common Packet Header: A common packet header (as common header format) where packet type is set to 0b00.

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).

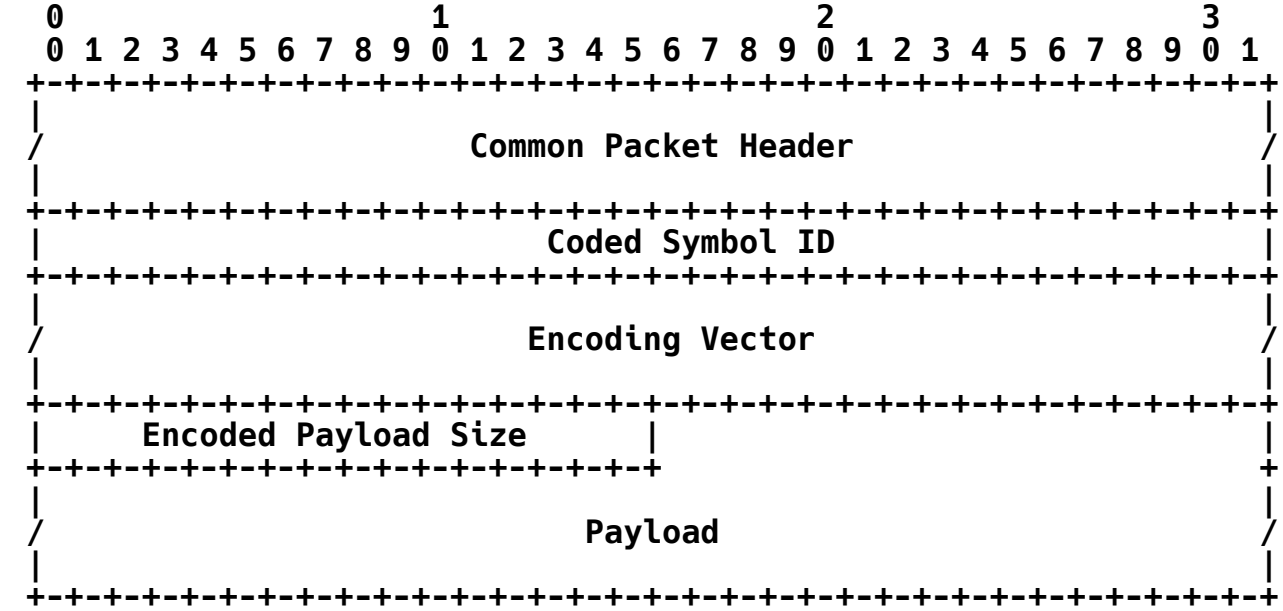


Figure 5: Coded Packet Format

Common Packet Header: A common packet header (as common header format) where packet type is set to 0b01.

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.

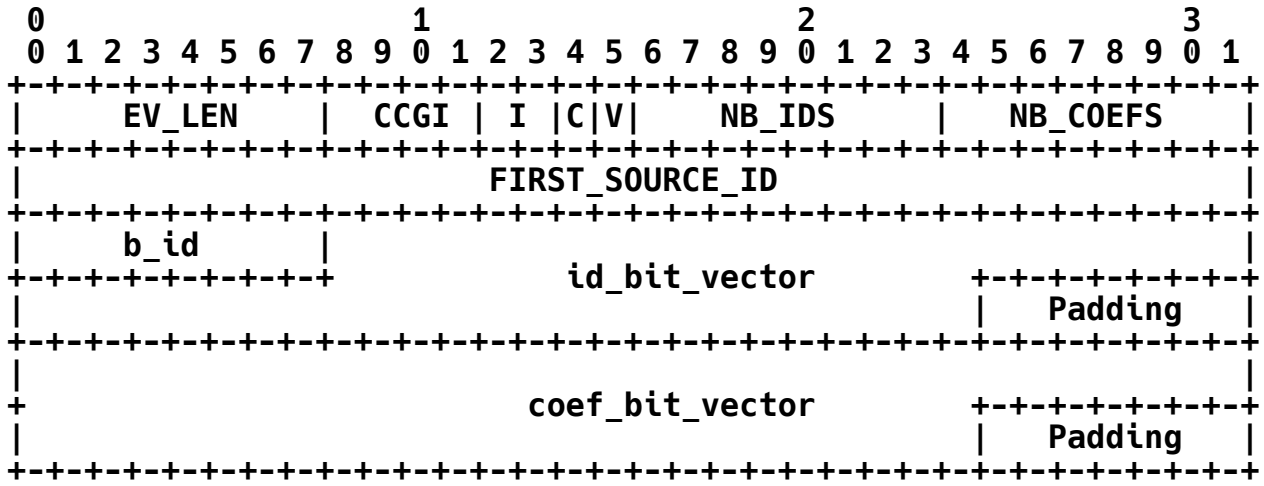


Figure 6: Encoding Vector Format

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

Coding Coefficient Generator Identifier (CCGI): 4-bit ID to identify the algorithm or function used to generate the coefficients. As a CCGI is included in each encoded vector, it MAY dynamically change between the generation of two 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 with 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

- **0b00: 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.**

- **0b01: 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, and 4 using CCGI set to 0b01. The coefficients will be alpha^((1 * 1) % 256), alpha^((1 * 2) % 256), and alpha^((1 * 4) % 256).

Store the Source Symbol ID Format (I) (2 bits):

* 0b00 means there is no source symbol ID information.

* 0b01 means the encoding vector contains the edge blocks of the source symbol IDs without compression.

* 0b10 means the encoding vector contains the compressed list of the source symbol IDs.

* 0b11 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 0b01 if the combination that 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 is set to 0b01, store the edge blocks as b_id * (NB_IDS * 2 - 1). If I is set to 0b10, store the edge blocks in a compressed way.

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 that is a multiple of 32 bits (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 in the list MAY be successive or not. If they are successive, the boundaries are stored in the encoding vector; it just needs 2*32 bits 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 that 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 symbol 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 the list of 32-bit unsigned integers (1, 3, 4, 5, 6, 10) into id_bit_vectors.

* If the source symbol IDs are stored as a list with compression, the field I MUST be set to 0b10. In this case, see 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 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 symbol 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]) that 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, it is 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 window update packets back to another building block. 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; hence, they could be omitted or lost in transmission without impacting the protocol behavior.

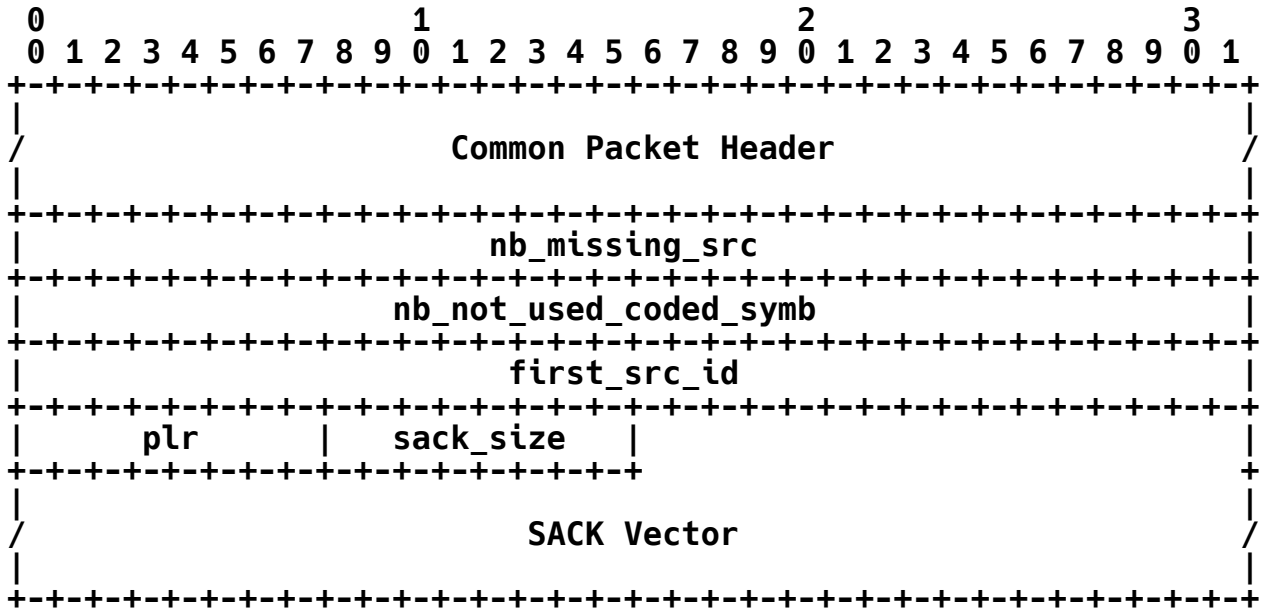


Figure 7: Window Update Packet Format

Common Packet Header: A common packet header (as common header format) where packet type is set to 0b10.

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 two unknown source symbols).

first_src_id: ID of the first source symbol to consider in the selective acknowledgment (SACK) vector.

plr: Packet loss ratio expressed as a percentage normalized to an 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 a 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 a value of 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 (i.e., in the case of a burst of losses) where it is better to drop and abandon some packets and 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 the 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 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. There are different research questions related to each of these scenarios that should be investigated for future protocol improvements. We summarize them in the following subsections.

6.1. Interaction with Congestion Control

The Tetrys and congestion control components generate two separate channels (see [RFC9265], 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 before 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.

The following topics, which are identified and discussed by [RFC9265], are adapted to the particular deployment cases of Tetrys (i.e., above, within, or below the transport layer):

* 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 congestion control efficiency. An approach is suggested to avoid hiding such signals in [RFC9265], Section 5.

* Tetrys and non-Tetrys flows sharing the same network links can raise fairness issues between these flows. In particular, the situation depends on whether some of these flows and not others are congestion controlled 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 multipath transmissions, with 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 similar 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 split these strategies into two distinct classes: Tetrys deployment inside the transport layer versus outside the transport layer (i.e., above or below). A deployment within the transport layer means that interactions between transport protocol mechanisms such as error recovery, congestion control, and/or flow control are envisioned. Otherwise, deploying Tetrys within a transport protocol that is not congestion controlled, 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 Section 4 of [RFC9265], 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 from consuming the whole available capacity as discussed in [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 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 with Tetrys has been proposed [THAI]. 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 it is congestion controlled or not; and

* 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 raises research questions when Tetrys is used to recover from IP datagram losses while tunneling. Applying redundancy without flow differentiation may contradict the service requirements of individual flows: some flows may be penalized more by high latency and jitter than by partial reliability, while other flows may be penalized more by partial reliability. In practice, head-of-line blocking impacts all flows in a similar manner despite their different needs, which indicates that more elaborate strategies inside Tetrys are needed.

7. Security Considerations

First of all, it must be clear that the use of FEC protection on a data stream does not provide any kind of security per se. On the contrary, the use of FEC protection on a data stream raises security risks. The situation with Tetrys is mostly similar to that of other content delivery protocols making use of FEC protection; this is well described in FECFRAME [RFC6363]. This section builds on this reference, adding new considerations to comply with Tetrys specificities when meaningful.

7.1. Problem Statement

An attacker can either target the content, protocol, or network. The consequences will largely differ reflecting various types of goals, like gaining access to confidential content, corrupting the content, compromising the Tetrys encoder and/or Tetrys decoder, or compromising the network behavior. In particular, several of these attacks aim at creating a Denial-of-Service (DoS) with consequences that may be limited to a single node (e.g., the Tetrys decoder), or

that may impact all the nodes attached to the targeted network (e.g., by making flows unresponsive to congestion signals).

In the following sections, we discuss these attacks, according to the component targeted by the attacker.

7.2. Attacks against the Data Flow

An attacker may want to access confidential content by eavesdropping the traffic between the Tetrys encoder/decoder. Traffic encryption is the usual approach to mitigate this risk, and this encryption can be applied to the source flow upstream of the Tetrys encoder or to the output packets downstream of the Tetrys encoder. The choice on where to apply encryption depends on various criteria, in particular the attacker model (e.g., when encryption happens below Tetrys, the security risk is assumed to be on the interconnection network).

An attacker may also want to corrupt the content (e.g., by injecting forged or modified source and coded packets to prevent the Tetrys decoder from recovering the original source flow). Content integrity and source authentication services at the packet level are then needed to mitigate this risk. Here, these services need to be provided below Tetrys in order to enable the receiver to drop undesired packets and only transfer legitimate packets to the Tetrys decoder. It should be noted that forging or modifying feedback packets will not corrupt the content, although it will certainly compromise Tetrys operation (see Section 7.3).

7.3. Attacks against Signaling

Attacks on signaling information (e.g., by forging or modifying feedback packets to falsify the good reception or recovery of source content) can easily prevent the Tetrys decoder from recovering the source flow, thereby creating a DoS. In order to prevent this type of attack, content integrity and source authentication services at the packet level are needed for the feedback flow from the Tetrys decoder to the Tetrys encoder as well. These services need to be provided below Tetrys in order to drop undesired packets and only transfer legitimate feedback packets to the Tetrys encoder.

Conversely, an attacker in position to selectively drop feedback packets (instead of modifying them) will not severely impact the function of Tetrys since it is naturally robust when challenged with such losses. However, it will have side impacts, such as the use of bigger linear systems (since the Tetrys encoder cannot remove well-received or decoded source packets from its linear system), which mechanically increases computational costs on both sides (encoder and decoder).

7.4. Attacks against the Network

Tetrys can react to congestion signals (Section 6.1) in order to provide a certain level of fairness with other flows on a shared network. This ability could be exploited by an attacker to create or reinforce congestion events (e.g., by forging or modifying feedback packets) that can potentially impact a significant number of nodes

attached to the network. In order to mitigate the risk, content integrity and source authentication services at the packet level are needed to enable the receiver to drop undesired packets and only transfer legitimate packets to the Tetrys encoder and decoder.

7.5. Baseline Security Operation

Tetrys can benefit from an IPsec / Encapsulating Security Payload (IPsec/ESP) [RFC4303] that provides confidentiality, origin authentication, integrity, and anti-replay services in particular. IPsec/ESP can be used to protect the Tetrys data flows (both directions) against attackers located within the interconnection network or attackers in position to eavesdrop traffic, inject forged traffic, or replay legitimate traffic.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. 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>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <https://www.rfc-editor.org/info/rfc5052>.

[RFC5445] Watson, M., "Basic Forward Error Correction (FEC) Schemes", RFC 5445, DOI 10.17487/RFC5445, March 2009, <https://www.rfc-editor.org/info/rfc5445>.

[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>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <https://www.rfc-editor.org/info/rfc6363>.

[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., 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>.

[RFC9265] Kuhn, N., Lochin, E., Michel, F., and M. Welzl, "Forward Erasure Correction (FEC) Coding and Congestion Control in Transport", RFC 9265, DOI 10.17487/RFC9265, July 2022, <https://www.rfc-editor.org/info/rfc9265>.

9.2. Informative References

[A-FEC] Bolot, J., Fosse-Parisis, S., and D. Towsley, "Adaptive FEC-based error control for Internet telephony", IEEE INFOCOM '99, Conference on Computer Communications, New York, NY, USA, Vol. 3, pp. 1453-1460, DOI 10.1109/INFCOM.1999.752166, March 1999, <https://doi.org/10.1109/INFCOM.1999.752166>.

[AHL-00] Ahlswede, R., Cai, N., Li, S., and R. Yeung, "Network information flow", IEEE Transactions on Information Theory, Vol. 46, Issue 4, pp. 1204-1216, DOI 10.1109/18.850663, July 2000, <https://doi.org/10.1109/18.850663>.

[CTCP] Kim, M., Cloud, J., ParandehGheibi, A., Urbina, L., Fouli, K., Leith, D., and M. Medard, "Network Coded TCP (CTCP)", arXiv 1212.2291v3, April 2013, <https://arxiv.org/abs/1212.2291>.

[RED-FEC] Lin, C., Shieh, C., Chilamkurti, N., Ke, C., and W. Hwang, "A RED-FEC Mechanism for Video Transmission Over WLANs", IEEE Transactions on Broadcasting, Vol. 54, Issue 3, pp. 517-524, DOI 10.1109/TBC.2008.2001713, September 2008, <https://doi.org/10.1109/TBC.2008.2001713>.

[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://datatracker.ietf.org/doc/html/draft-singh-rmcat-adaptive-fec-03>.

[Tetrys] Lacan, J. and E. Lochin, "Rethinking reliability for long-delay networks", International Workshop on Satellite and Space Communications, Toulouse, France, pp. 90-94, DOI 10.1109/IWSSC.2008.4656755, October 2008, <https://doi.org/10.1109/IWSSC.2008.4656755>.

[Tetrys-RT] Tournoux, P., 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, pp. 797-812, DOI 10.1109/TMM.2011.2126564, August 2011, <http://dx.doi.org/10.1109/TMM.2011.2126564>.

[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 Issue 4, pp. 449-461, DOI 10.1016/j.image.2014.02.003, April 2014, <https://doi.org/10.1016/j.image.2014.02.003>.

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.

Authors' Addresses

Jonathan Detchart
ISAE-SUPAERO
BP 54032
10, avenue Edouard Belin
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
BP 54032
10, avenue Edouard Belin

31055 Toulouse CEDEX 4
France
Email: jerome.lacan@isae-supaero.fr

Vincent Roca
INRIA
Inovallee; Montbonnot
655, avenue de l'Europe
38334 St Ismier CEDEX
France
Email: vincent.roca@inria.fr