Workgroup: NWCRG
Internet-Draft: draft-irtf-nwcrg-tetrys-00
Published: 17 December 2021
Intended Status: Experimental
Expires: 20 June 2022
Authors: J. Detchart E. Lochin J. Lacan V. Roca
ISAE-SUPAERO ENAC ISAE-SUPAERO INRIA

Tetrys, an On-the-Fly Network Coding protocol

Abstract

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

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 an 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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 20 June 2022.

include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

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 [RFC3452] and the FECFRAME [RFC8680] working groups. The protocol presented here can be

seen as a network coding extension to standards solutions. The current proposal can be considered 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. 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 window. The size of this window can be fixed or dynamically updated. If the window is full, incoming source packets 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.

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 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 with a transmission. Compared to FEC block codes, this allows reducing 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

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 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 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

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

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 can be a multiplex of several upper layer streams. Transmission can be over a single path or multiple paths. Besides, the flow can be sent in unicast, multicast, or anycast mode. This is the simplest use-case,

that is very much aligned with currently proposed scenarios for end-to-end streaming.

3.2. Overview

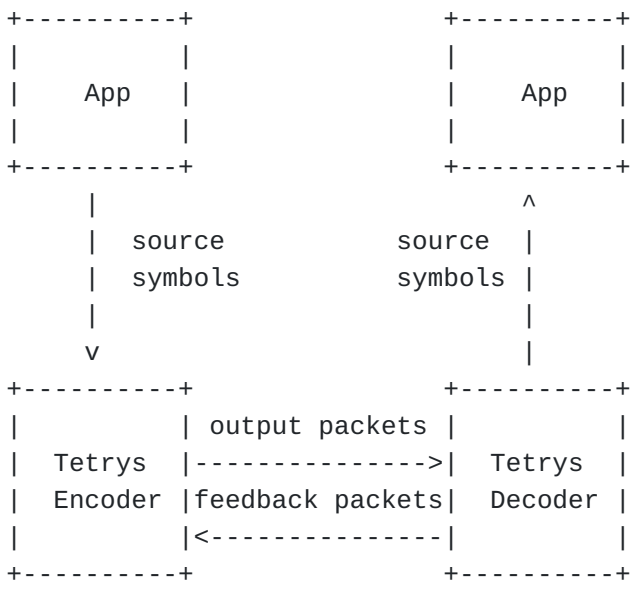

Figure 1: Tetrys Architecture

The Tetrys protocol features several key functionalities. The mandatory features 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, in a manner that was previously used in FEC;
- 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 Section 7.1 for further details;

Several building blocks provide these functionalities:

- *The Tetrys Building Block: this BB 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 can be used.

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

- *Other ?

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. Packet Format

4.1. Common Header Format

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

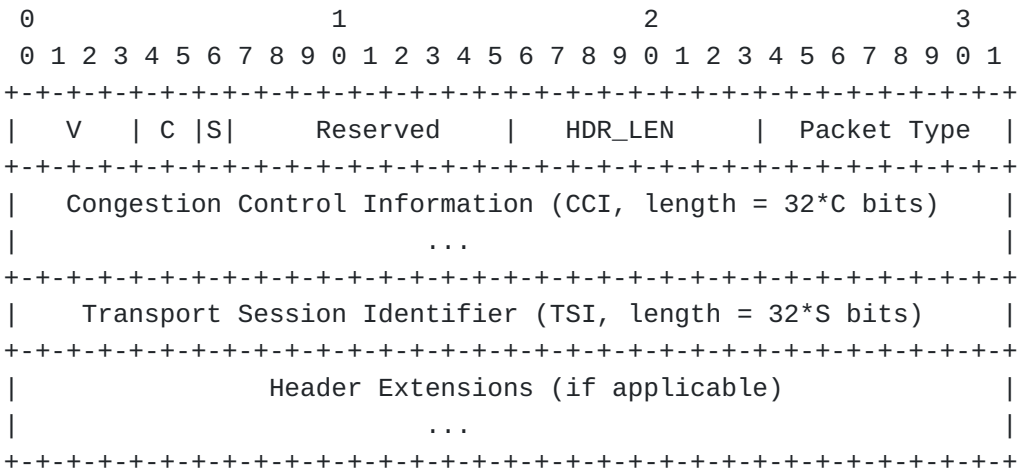

Figure 2: Common Header Format

As already noted above in the document, this format is compatible with LCT 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 can 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 other headers or packet payload.

*Packet Type: 8 bits. Type of packet.

*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 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.

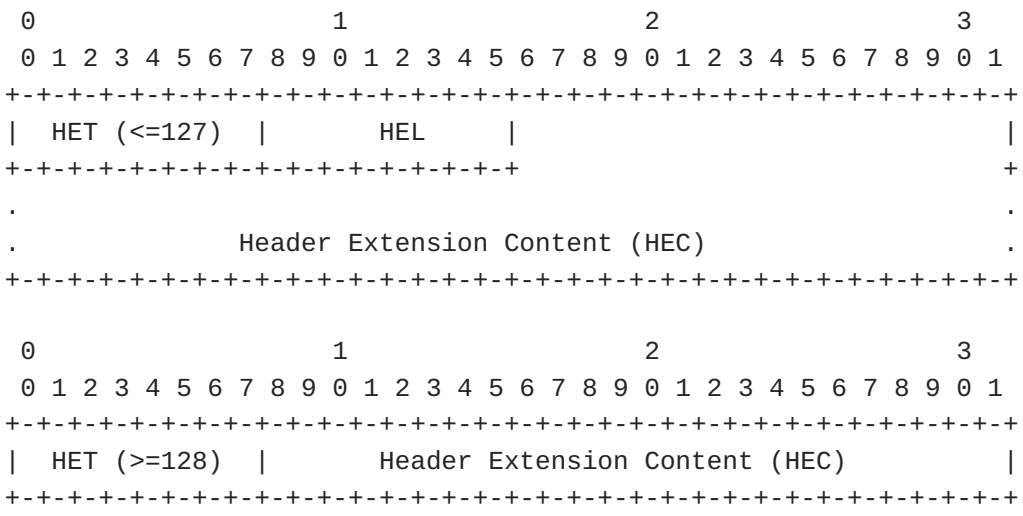

Figure 3: Header Extension Format

*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 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 a Common Packet Header encapsulation, a Source Symbol ID and a source symbol (payload). The source symbols can have variable sizes.

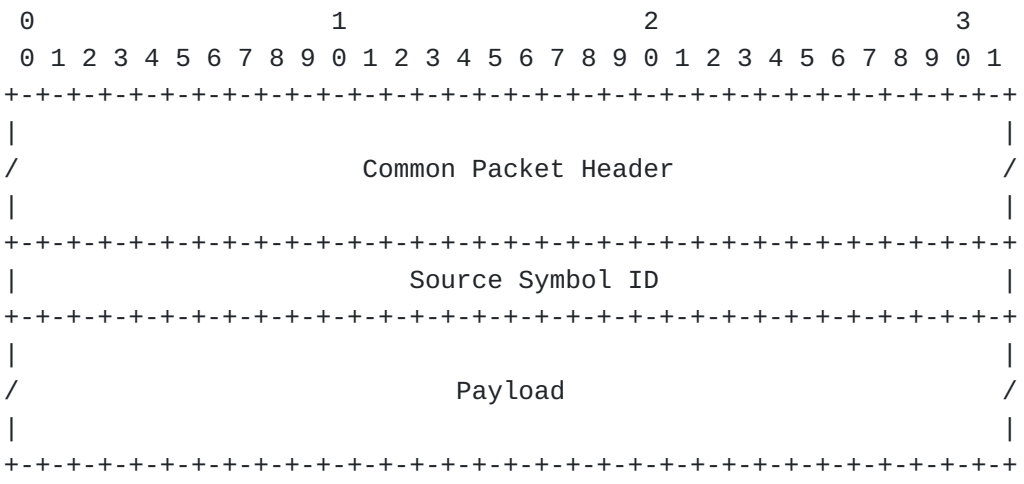

Figure 4: Source Packet Format

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. 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).

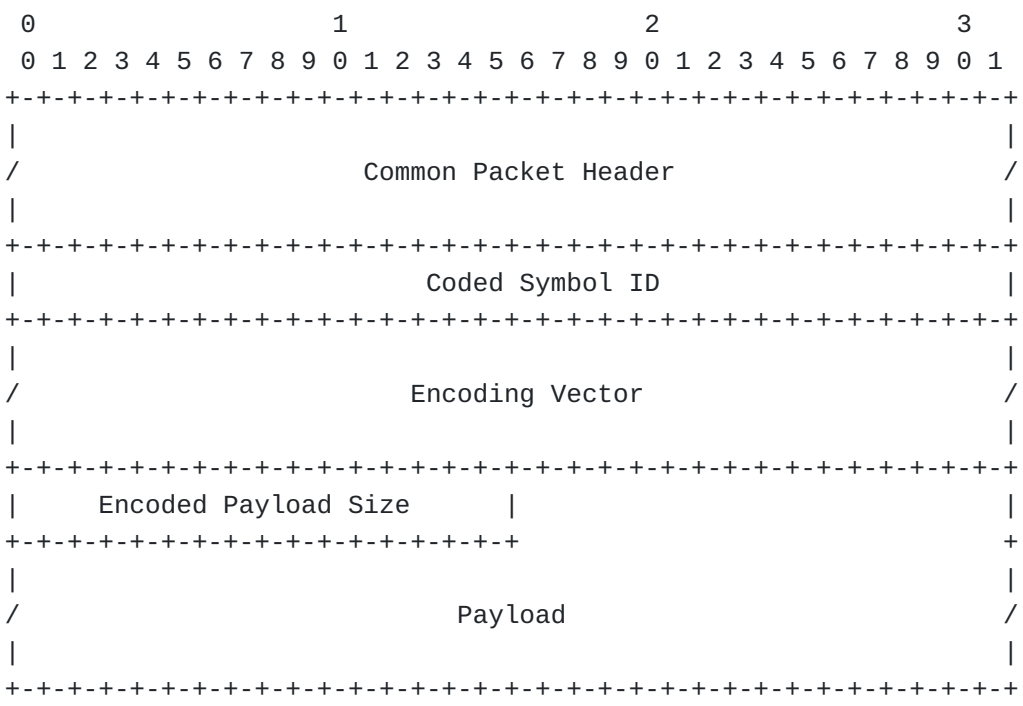

Figure 5: Coded Packet Format

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 6.1.1.2)).

Payload: the coded symbol.

4.4. Acknowledgement Packet Format

A Tetrys Decoding 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 acknowledgment packets are OPTIONAL hence they could be omitted or lost in transmission without impacting the protocol behavior.

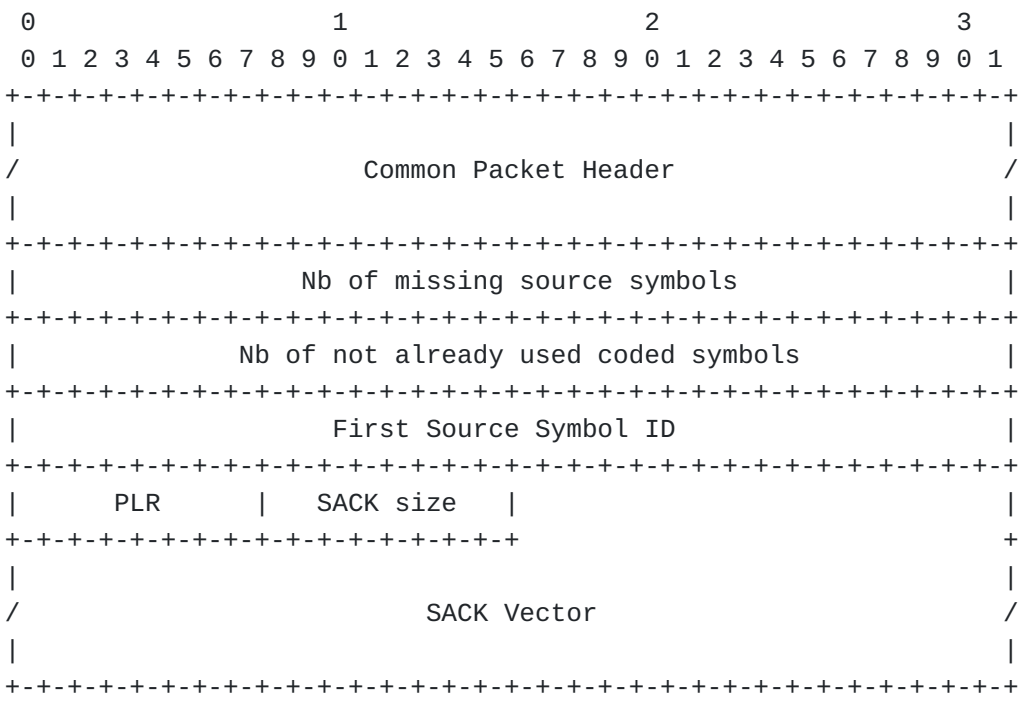

Figure 6: Acknowledgement Packet Format

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

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). This value is used in the case of dynamic code rate or for statistical purpose. The choice of calculation is left to the appreciation of the developer 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 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 the i-th position means that this acknowledgment packet acknowledges the source symbol of ID equal to "First Source Symbol ID" + i.

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 or decoders.

5.2. Table of Identifiers

0000: Vandermonde based coefficients over a finite field with 2^^4 elements,defined by the primitive polynomial 1+x+x^^4. Each coefficient is built as alpha^((source_symbol_id*coded-symbol_id) % 16), with alpha the root of the primitive polynomial.

0001: Vandermonde based coefficients over a finite field with 2^^8 elements,defined by the primitive polynomial 1+x^^2+x^^3+x^^4+x^^8. Each coefficient is built as alpha^((source_symbol_id*coded-symbol_id) % 256), with alpha the root of the primitive polynomial.

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 the 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 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 that are not yet acknowledged by the receiver.

A Tetrys Encoding Building Block 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 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 typical example of such a 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 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, the boundaries are 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 * [(2 * NB blocks) - 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 blocks 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.

If the source IDs are stored, the number of blocks MUST be different from 0.

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

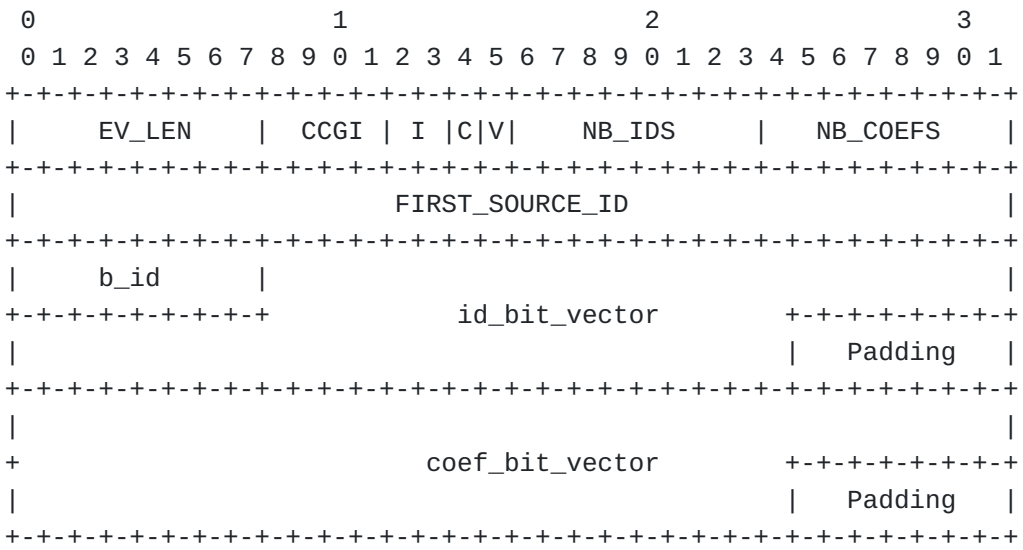

Figure 7: Encoding Vector Format

*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 (see Section 5). As a CCGI is included in each encoded vector, it can dynamically change between the generation of 2 coded symbols.

*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.

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

*Having source symbols with variable size (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.

*Number of IDs used to store the source symbol IDs (NB_IDS): the number of IDs stored (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 (see Section 6.1.1.1).

*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 coeffecient).

*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 on 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 sends periodic feedback indicating the received or decoded source symbols. In the case of 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:

*If the acknowledgment 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.

*If the acknowledgment 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. Decoding

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

7. Research Issues

The design of Tetrys protocol presented in this document provides the baseline allowing communication between a Tetrys encoder and a Tetrys decoder. At this stage, the detailed specifications only focus on the coding and decoding aspects. The objective of this document is first to provide guidelines to implement Tetrys as a standalone protocol or to embed Tetrys inside an existing protocol at the application layer or the IP layer. However, both cases raise manifold research efforts to come up with a complete protocol specification. Despite mandatory communication protocol operations such as opening/closing procedures and timeout/reset, we identified the following research issues that would need further discussion.

7.1. Interaction with Existing Congestion-Controlled Transport Protocol

Tetrys coding and congestion control can be seen as two separate channels. In practice, implementations may mix the signals exchanged on these channels. This raises several concerns that must be tackled when considering using Tetrys conjointly with a congestion-controlled transport protocol. All these numerous research issues are discussed in a separate document [I-D.irtf-nwcrg-coding-and-congestion] . In particular, this document investigates end-to-end unicast data transfer with FEC coding in the application (above the transport), within the transport, or directly below the transport; the relationship between transport layer and application requirements; and the case of transport multipath and multi-streams applications.

7.2. Adaptive Coding Rate

In a particular context, a redundancy adaptation algorithm might be considered helpful or mandatory when the network condition (e.g., delay, loss rate) strongly varies over time. Hence, it requires an enhanced mechanism for erasure codes to adapt to network dynamics similarly to [A-FEC] . However, the dynamic adaptation of an on-the-fly coding rate is slightly more complex than a block code. Furthermore, this adaptation can be done conjointly 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. In brief, 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. However, this kind of algorithms should deserve more research to be deployed in practice.

7.3. Using Tetrys Above The IP Layer For Tunneling

The use of Tetrys to protect from losses an aggregate of flows raise various issues. This occurs when an encoding mechanism is enabled below the IP layer and builds redundancy without flows differentiation. This is typically the case in a tunnel. The main problem relates to head-of-line blocking when decoding multiple flows. The number of source packets might vary following their own loss probability and lead to decoding blocking in waiting for source data packets to be suppressed from a given repair packet. This kind of issue could lead to a decrease of the decoding performance and should be further investigated. Note this research issue joins the topics discussed in the IRTF LOOPS working group [I-D.li-tsvwg-loops-problem-opportunities] .

8. Security Considerations

Tetrys inherits a subset of the security issues described as those 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.

9. Privacy Considerations

N/A

10. IANA Considerations

N/A

11. Acknowledgments

First, the authors want to sincerely 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.

12. References

12.1. 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>.

[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>.

[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>.

[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>.

12.2. 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.

[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-09, 25 June 2021, <https://www.ietf.org/archive/id/draft-irtf-nwcrg-coding-and-congestion-09.txt>.

[I-D.li-tsvwg-loops-problem-opportunities]
Li, Y., Zhou, X., Boucadair, M., Wang, J., and F. Qin, "LOOPS (Localized Optimizations on Path Segments) Problem Statement and Opportunities for Network-Assisted Performance Enhancement", Work in Progress, Internet-Draft, draft-li-tsvwg-loops-problem-opportunities-06, 13 July 2020, <https://www.ietf.org/archive/id/draft-li-tsvwg-loops-problem-opportunities-06.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.

[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