

NWCRG  
Internet-Draft  
Intended status: Experimental  
Expires: March 12, 2020

J. Heide  
Steinwurf Aps  
S. Shi  
K. Fouli  
M. Medard  
Code On Network Coding LLC  
V. Chook  
Inmarsat PLC  
September 09, 2019

Random Linear Network Coding (RLNC)-Based Symbol Representation  
draft-heide-nwcrg-rlnc-03

Abstract

This document describes a symbol representation for Random Linear Network Coding (RLNC) schemes used for reliable data transfer. Specifically, the following features are discussed and incorporated: both block RLNC and a sliding window RLNC, varying data frame sizes, and one or multiple symbols associated with a single symbol representation header.

Requirements Language

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 RFC 2119 [RFC2119].

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 March 12, 2020.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Symbol Representation . . . . .	3
2.1. Representation Setup . . . . .	4
2.2. Field Types and Formats . . . . .	5
2.3. Externally Specified Parameters Required . . . . .	7
2.4. Small Encoding Window . . . . .	7
2.4.1. Examples . . . . .	9
2.5. Large Encoding Window . . . . .	10
3. Security Considerations . . . . .	11
4. IANA Considerations . . . . .	11
5. References . . . . .	11
5.1. Normative References . . . . .	11
5.2. Informative References . . . . .	12
Authors' Addresses . . . . .	12

## 1. Introduction

Symbol representation specifies the format of the symbol-carrying data unit that is to be used in network coding operations, including header format and symbol concatenation. This document describes a symbol representation format intended to be used for Network Coding in general, and for Random Linear Network Coding (RLNC) in particular [HK03].

Owing to its dynamic structure, network coding has requirements that are distinct from conventional point-to-point codes, leading to a highly reconfigurable symbol set. Consequently, the design choices related to symbol representation are particularly important in network coding as they have a direct impact on the viability of network protocols, topologies, and architecture [RLNC-Background]. For example, recoding [RLNC-Background] requires the coefficients to

be accessible at the recoding nodes. Hence, architectures and protocols requiring recoding must specify coefficient location in their symbol representation.

In addition to providing background on RLNC, [RLNC-Background] argues that careful design and specification of a symbol representation is a requirement for any viable network coding protocol, architecture, or topology.

## 2. Symbol Representation

This section provides a symbol representation design for implementing RLNC-based erasure correction schemes. In the described symbol representation design, multiple symbols are concatenated and associated with a single symbol representation header.

The symbol representation design is provided for constructing a data payload portion of a data packet for a protocol that utilizes a generation-based or sliding-window RLNC, where recoding can be used at intermediate nodes. A data packet data payload comprises one or more symbol representations. Each symbol representation in turn comprises one or more symbols that can be systematic, coded or recoded. The use of this symbol representation design is not limited by transmission schemes. It can be applied to unicast, multiple-unicast, multicast, multi-path, and multi-source settings and the like.

Coding coefficient vectors must be implicitly or explicitly transmitted from the sender to the receiver, generally along with the coded data for successful decoding of the original data. One option is to attach each coding coefficient vector to the corresponding coded symbol as a header, thus also enabling recoding at intermediate nodes. Another option is to attach the current state of a pseudo-random generator for generating the coding coefficient vector, to reduce the size of the header. Adding a header to each symbol may result in a high overhead when the symbol size is small or when generation or sliding window size is large. Adding a joint header to the beginning of each generation may also cause synchronization to be re-initiated only at the beginning of each generation instead of every symbol. In what follows, a symbol representation is provided that allow for both of these options such that both a general representation with coding coefficients and a compact representation with a seed for generating the coding coefficients can be used, in order to reduce the header overhead.

## 2.1. Representation Setup

This section specifies a symbol representation that enables both a general form with coding coefficient vectors attached, and a compact form where a seed is attached which is used to generate one or multiple coding coefficient vectors. Different maximum GENERATION and WINDOW SIZE are supported for RLNC encoding, recoding, and decoding.

To encode over a set of data symbols, a coding coefficient vector is first generated, comprising a number of finite field elements as specified by a GENERATION SIZE or WINDOW SIZE variable. For a generation based code the GENERATION SIZE defines the number of original symbols in each generation. For a window based code the WINDOW SIZE specifies the maximal number of symbols in the window over which coding can be performed. In the case of systematic codes, systematic symbols correspond to unit coding coefficient vectors.

Figure 1 illustrates the general symbol representation design. Four header fields precede the symbol data: TYPE flag (T), SYMBOLS, ENCODER RANK, and SEED or CODING COEFFICIENTS. The TYPE Flag (T) indicates if the symbol is systematic, coded, or recoded. SYMBOLS indicates the number of symbols in the SYMBOL(S) DATA field. ENCODER RANK represents the current rank of the encoder, which is the number of symbols being linearly combined. SEED is used to generate the coding coefficient vector(s) using a pseudo-random number generator, for a compact form of the symbol representation. The CODING COEFFICIENTS field is a list of SYMBOLS number of coding coefficient vectors used to generate the SYMBOL(S) DATA, and used in the case where no random number generator is available or practical

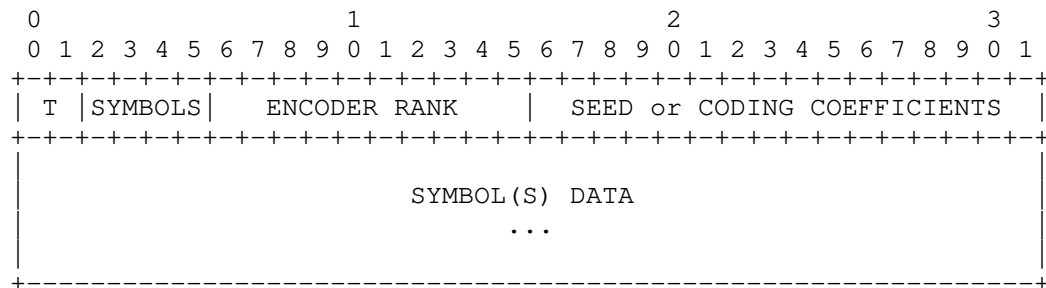


Figure 1: A general symbol representation design.

## 2.2. Field Types and Formats

The TYPE Flag (T) indicates if the symbol is systematic, coded, or recoded, and has the following properties:

- o 2 bits long.
- o If the TYPE flag is '1', all symbols included in this symbol representation are systematic or uncoded, with symbol index starting from ENCODER RANK. This option allows for efficient representation of systematic symbols.
- o If the TYPE is '2', all symbols included in this symbol representation are coded, with coding coefficient vectors generated using the included SEED and the ENCODER RANK. Consequently, only the first ENCODER RANK elements in the coding coefficient vector can be non-zero, whereas the remaining elements (e.g. GENERATION SIZE - ENCODER RANK) in the coding coefficient vector are zeros. This option allows for compact and efficient representation of coded symbols, which may also subsequently be recoded.
- o If the TYPE is '3', all symbols included in this symbol representation are either uncoded, coded or recoded. Each coding vector included is composed of GENERATION SIZE or WINDOW SIZE coefficients.

SYMBOLS indicates the number of symbols in the 'Symbol(s) Data' field, and has the following properties:

- o 4 bits long. A maximum number of 15 symbols are concatenated within each symbol representation.
- o The special case of SYMBOLS = 0 indicates that zero symbols are included, and consequently the size of SYMBOLS(S) DATA is 0 bytes. This can, for example, be used to implement a flush functionality or ensure that protocol operations do not stop in certain case for purely event-driven protocols.

ENCODER RANK represents the current rank of the encoder, that is, the number of original symbols used to compute the coded symbols(s). It has the following properties:

- o MUST be no larger than GENERATION/WINDOW SIZE.
- o If TYPE flag is '1', ENCODER RANK is the symbol index of the first data symbol in this symbol representation.

- o If TYPE flag is '2' or '3', ENCODER RANK is the number of data symbols over which coding was performed for all coded symbols in this symbol representation.

SEED is used to generate the coding coefficient vector(s) using a pseudo-random number generator, for a compact form of the symbol representation, and has the following properties:

- o The SEED field is only present when TYPE flag is '2'. If TYPE is '1' or '3', this field is absent.
- o The pseudo-random generator MUST be seeded with this value and all coding coefficient vectors are produced by the same generator. For example, if ENCODER RANK is 12, then the coding coefficient vector for the first symbol in this symbol representation is coefficients 0 through 11 generated by the pseudo-random generator seeded by SEED, and coding coefficient vector for the second symbol in this symbol representation is coefficients 12 through 23 generated by the pseudo-random generator seeded by SEED. If GENERATION/WINDOW SIZE is larger than ENCODER RANK, the remaining coefficients in the coding coefficient vector are zero.
- o To ensure that SEED can be interpreted correctly at the receiver, the same pseudo-random number generator MUST be used by the sender and a recoding or receiving node. Otherwise, more than one SEED field would need to be used.
- o 8 bits long. Thus, 256 different seed values can be served. One SEED is used per symbol representation, each of which can contain up to 15 symbols, all derived using the same SEED. For distinct ENCODER RANKs, different coding coefficient vectors would be generated from the same SEED, since only an ENCODER RANK number of coefficients from the random generator is grouped as a coding coefficient vector, before progressing to the next coding coefficient vector for the next symbol in the symbol representation. Consequently, the maximal number of coded symbols that can be generated for a generation is  $|SEED| * |SYMBOLS| * |ENCODER RANK|$  which in the best case is  $(2^8) * (2^4 - 1) * (2^{10}) \sim 2^{22}$ , which for all practical considerations can be considered as an infinite number of coded symbols. If all coded symbols that can be represented using a SEED is exhausted, symbols where the coding coefficient vectors is included can be sent instead.

CODING COEFFICIENTS field is a list of SYMBOLS number of coding vectors used to generate the ensuing SYMBOL(S) DATA, and has the following properties:

- o The CODING COEFFICIENT field is only present when TYPE flag is '3'. If TYPE is '1' or '2', this field is absent.
- o Each coding coefficient vector includes ENCODER RANK number of coding coefficients.

### 2.3. Externally Specified Parameters Required

This section specifies parameters that are REQUIRED for the use of this symbol representation but which are not included in the symbol representation and therefore MUST be communicated by means of some outer mechanism. Typically these parameters will be static throughout a protocol session. Consequently, there is little to gain by incorporating these parameters into the representation but conversely it would add additional overhead.

- o Field polynomial, the underlying field over which coding is performed.
- o Pseudo Random Generator, used to generate coding coefficient vectors.
- o Symbol Size, used to divide the original data into symbols.
- o GENERATION SIZE or WINDOW SIZE, for block and sliding window codes, respectively.
- o Small or large encoding window, this symbol representation supports both a small and a large coding window, but the variant used is not communicated.

### 2.4. Small Encoding Window

In a first small encoding window symbol representation, ENCODER RANK is 10 bits long, and the maximum GENERATION/WINDOW SIZE is  $2^{10}$ .

Figures 2 to 4 below illustrate systematic, coded, and recoded symbol representations within an encoding window of size  $2^{10}$ . Systematic symbols are uncoded. Coded symbols are compact in form and comprise a seed for coding coefficient generation. Recoded symbols are general in form and comprise the coding coefficient vectors explicitly.

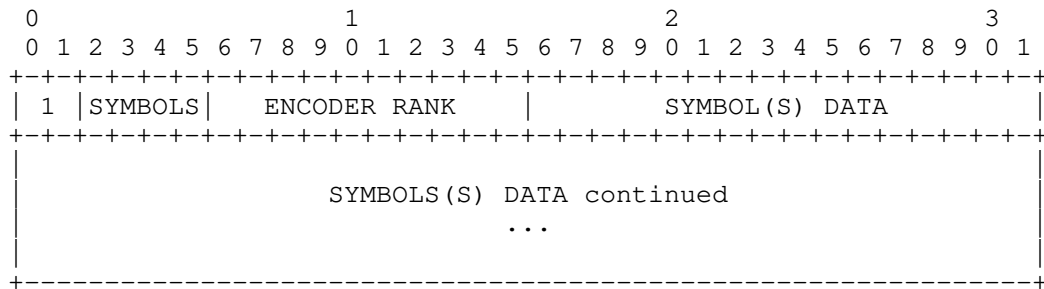


Figure 2: A systematic symbol representation.

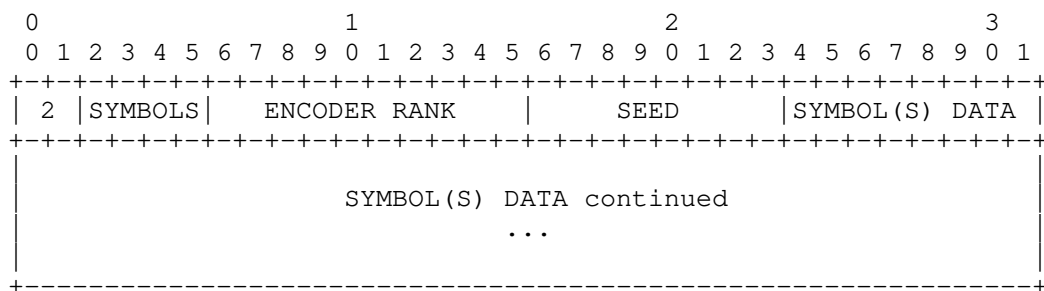


Figure 3: A compact, coded symbol representation.

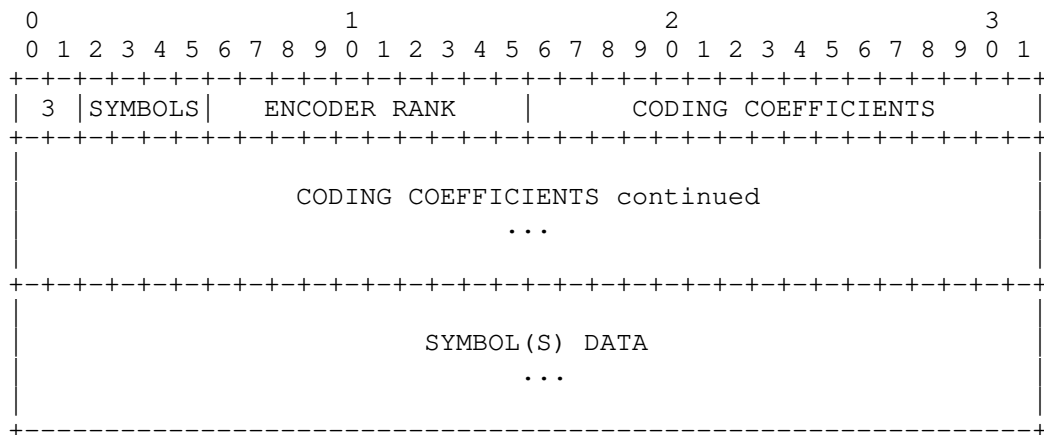


Figure 4: A recoded symbol representation.



### 2.4.1. Examples

The following examples show different symbol representations for an illustrative case where the symbol size is 2 bytes, GENERATION/WINDOW SIZE is 8, and field size is  $2^8$ .

Example 1: Three systematic symbols with ID 0, 1 and 2. As the TYPE flag is '1', SEED/CODING COEFFICIENTS is absent, and ENCODER RANK is the symbol index of the first data symbol with ID 0 in this compact symbol representation.

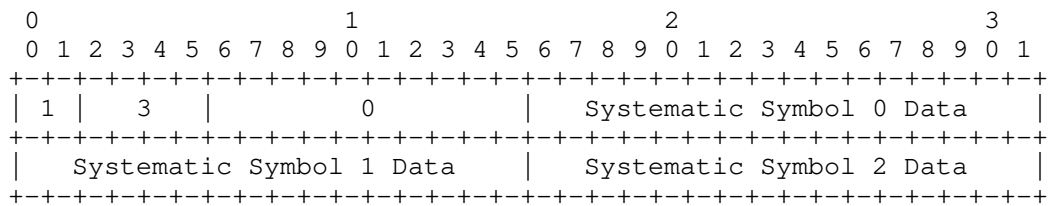


Figure 5: A symbol representation with 3 systematic, uncoded symbols.

Example 2: Two coded symbols using a compact representation. In this example, TYPE is '2', the SEED to the pseudo-random number generator shared by the sender and receiver is 4. The coding coefficient vector for Symbol A is coefficients 0 to 7 generated by the pseudo-random number generator, the coding coefficient vector for symbol B is coefficients 8 to 15 generated by the pseudo-random number generator.

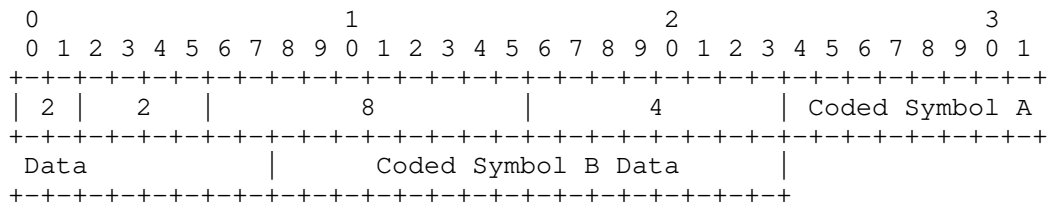


Figure 6: A symbol representation with 2 coded symbols.

Example 3: Two recoded symbols. Coefficients A0 to A7 constitute the coding coefficient vector for Symbol A, coefficients B0 to B7 constitute the coding coefficient vector for symbol B. In practical implementations, symbol sizes are much larger than 2, leading to amortization of the coding coefficient overheads.

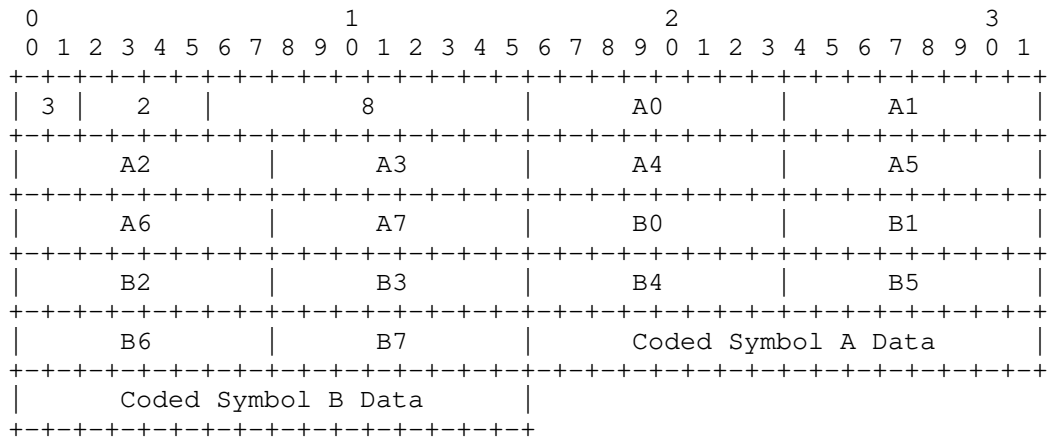


Figure 7: A symbol representation with 2 recoded symbols having coding coefficients attached.

## 2.5. Large Encoding Window

In a second large encoding window symbol representation, ENCODER RANK is 18-bit long, and the maximum GENERATION/WINDOW SIZE is  $2^{18}$ .

Figures 8 to 10 below illustrate systematic, coded, and recoded symbol representations within an encoding window of size  $2^{18}$ . Systematic symbols are uncoded. Coded symbols are compact in form and comprise a seed for coding coefficient generation. Recoded symbols are general in form and comprise the coding coefficient vectors explicitly (CODING COEFFICIENTS or CODING COEFFS).

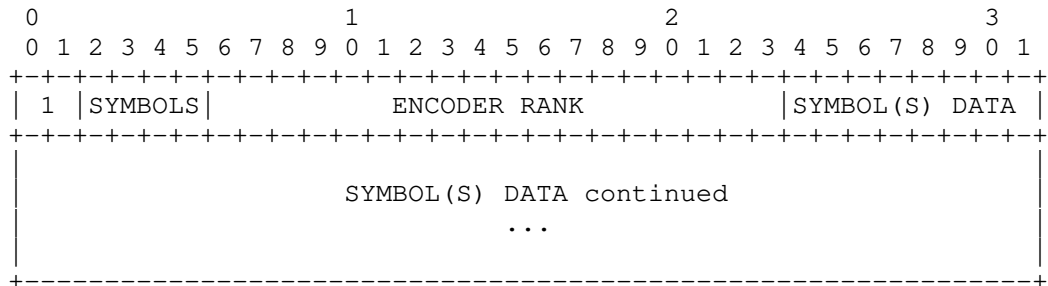


Figure 8: A systematic symbol representation.

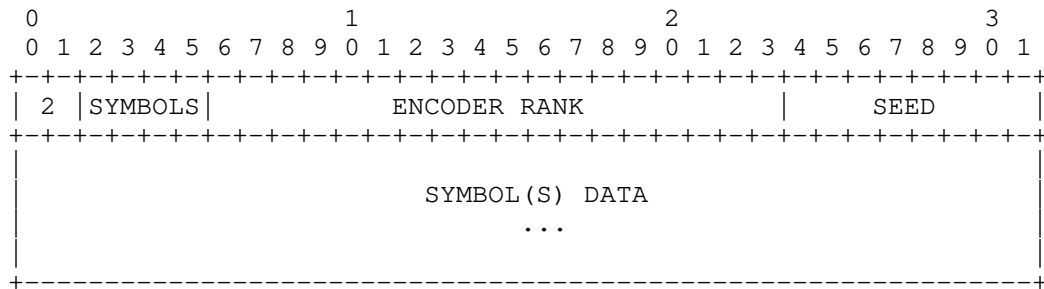


Figure 9: A coded symbol representation.

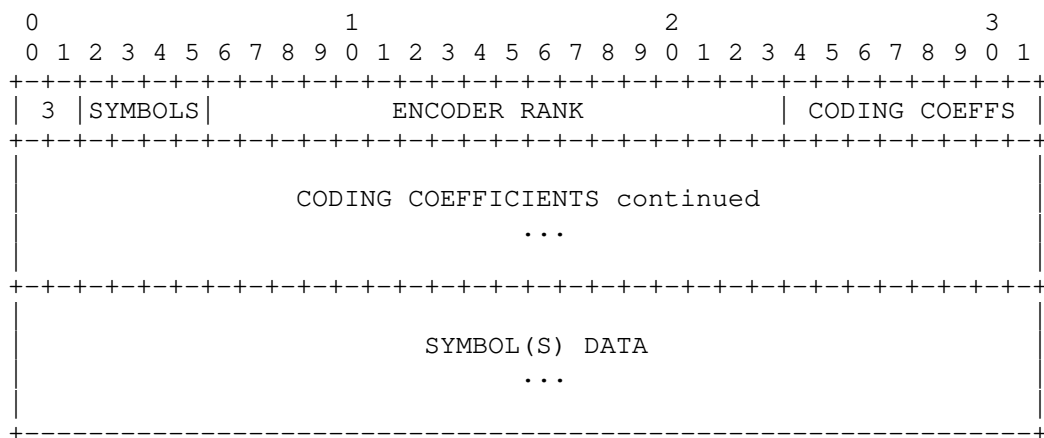


Figure 10: A recoded symbol representation.

### 3. Security Considerations

This document does not present new security considerations.

### 4. IANA Considerations

This document has no actions for IANA.

### 5. References

#### 5.1. 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>>.

[RLNC-Background]

Heide, J., Shi, S., Fouli, K., Medard, M., and V. Chook,  
"Random Linear Network Coding (RLNC): Background and  
Practical Considerations", February 2018,  
<[https://datatracker.ietf.org/doc/  
draft-heide-nwcrg-rlnc-background/](https://datatracker.ietf.org/doc/draft-heide-nwcrg-rlnc-background/)>.

## 5.2. Informative References

[HK03] Ho, T., Koetter, R., Medard, M., Karger, D., and M.  
Effros, "The Benefits of Coding over Routing in a  
Randomized Setting", July 2003,  
<<http://ieeexplore.ieee.org/document/1228459/>>.

## Authors' Addresses

Janus Heide  
Steinwurf Aps  
Aalborg  
Denmark

Email: [janus@steinwurf.com](mailto:janus@steinwurf.com)

Shirley Shi  
Code On Network Coding LLC  
Cambridge  
USA

Email: [xshi@alum.mit.edu](mailto:xshi@alum.mit.edu)

Kerim Fouli  
Code On Network Coding LLC  
Cambridge  
USA

Email: [fouli@codeontechnologies.com](mailto:fouli@codeontechnologies.com)

Muriel Medard  
Code On Network Coding LLC  
Cambridge  
USA

Email: [muriel.medard@codeontechnologies.com](mailto:muriel.medard@codeontechnologies.com)

Vince Chook  
Inmarsat PLC  
London  
United Kingdom

Email: Vince.Chook@inmarsat.com

NWCRG  
Internet-Draft  
Intended status: Informational  
Expires: August 15, 2019

J. Heide  
Steinwurf Aps  
S. Shi  
K. Fouli  
M. Medard  
Code On Network Coding LLC  
V. Chook  
Inmarsat PLC  
February 11, 2019

Random Linear Network Coding (RLNC): Background and Practical  
Considerations  
draft-heide-nwcrg-rlnc-background-00

Abstract

This document describes the use of Random Linear Network Coding (RLNC) schemes for reliable data transport. Both block and sliding window RLNC code implementations are described. By providing erasure correction using randomly generated repair symbols, such RLNC-based schemes offer advantages in accommodating varying frame sizes and dynamically changing connections, reducing the need for feedback, and lowering the amount of state information needed at the sender and receiver. The practical considerations' section identifies RLNC-encoded symbol representation as a valuable target for standardization.

Requirements Language

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 RFC 2119 [RFC2119].

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 August 15, 2019.

#### Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	2
1.1. Random Linear Network Coding (RLNC) Basics . . . . .	3
1.2. Generation-Based RLNC . . . . .	5
1.3. Sliding Window RLNC . . . . .	7
1.4. Recoding . . . . .	8
2. Practical Considerations . . . . .	9
2.1. Symbol Representation . . . . .	9
2.1.1. Symbol Representation as a Standardization Approach .	9
2.1.2. Coding Parameter Design Considerations . . . . .	12
3. Security Considerations . . . . .	13
4. IANA Considerations . . . . .	13
5. References . . . . .	13
5.1. Normative References . . . . .	13
5.2. Informative References . . . . .	14
Authors' Addresses . . . . .	14

#### 1. Introduction

Network Coding is a coding discipline that jointly improves network reliability and efficiency. In general communication networks, source coding is performed as a compression mechanism to reduce data redundancy and to reduce resources necessary for data transportation over the network. Channel coding, on the other hand, introduces redundancy for data transmission reliability over lossy channels. Network coding adds another layer of coding in-between these two. Random Linear Network Coding (RLNC) is an efficient network coding approach that enables network nodes to generate independently and randomly linear mappings of input to output data symbols over a finite field.

This document provides background on RLNC operation. The document also includes an open section for practical considerations where topics such as standardization and RLNC-encoded symbol representations are addressed.

### 1.1. Random Linear Network Coding (RLNC) Basics

Unlike conventional communication systems based on the "store-and-forward" principle, RLNC allows network nodes to independently and randomly combine input source data into coded symbols over a finite field [HK03]. Such an approach enables receivers to decode and recover the original source data as long as enough linearly independent coded symbols, with sufficient degrees of freedom, are received. At the sender, RLNC can introduce redundancy into data streams in a granular way. At the receiver, RLNC enables progressive decoding and reduces feedback necessary for retransmission. Collectively, RLNC provides network utilization and throughput improvements, high degrees of robustness and decentralization, reduces transmission latency, and simplifies feedback and state management.

To encode using RLNC, original source data are divided into symbols of a given size and linearly combined. Each symbol is multiplied with a scalar coding coefficient drawn randomly from a finite field, and the resulting coded symbol is of the same size as the original data symbols.

Thus, each RLNC encoding operation can be viewed as creating a linear equation in the data symbols, where the random scalar coding coefficients can be grouped and viewed as a coding vector. Similarly, the overall encoding process where multiple coded symbols are generated can be viewed as a system of linear equations with randomly generated coefficients. Any number of coded symbols can be generated from a set of data symbols, similarly to expandable forward error correction codes specified in [RFC5445] and [RFC3453]. Coding vectors must be implicitly or explicitly transmitted from the sender to the receiver for successful decoding of the original data. For example, sending a seed for generating pseudo-random coding coefficients can be considered as an implicit transmission of the coding vectors. In addition, while coding vectors are often transmitted together with coded data in the same data packet, it is also possible to separate the transmission of coding coefficient vectors from the coded data, if desired.

To reconstruct the original data from coded symbols, a network node collects a finite but sufficient number of degrees of freedom for solving the system of linear equations. This is beneficial over conventional approaches as the network node is no longer required to



gather each individual data symbol. In general, the network node needs to collect slightly more independent coded symbols than there are original data symbols, where the slight overhead arises because coding coefficients are drawn at random, with a non-zero probability that a coding vector is linearly dependent on another coding vector, and that one coded symbol is linearly dependent on another coded symbol. This overhead can be made arbitrarily small, provided that the finite field used is sufficiently large.

A unique advantage of RLNC is the ability to re-encode or "recode" without first decoding. Recoding can be performed jointly on existing coded symbols, partially decoded symbols, or uncoded systematic data symbols. This feature allows intermediate network nodes to re-encode and generate new linear combinations on the fly, thus increasing the likelihood of innovative transmissions to the receiver. Recoded symbols and recoded coefficient vectors have the same size as before and are indistinguishable from the original coded symbols and coefficient vectors.

In practical implementations of RLNC, the original source data are often divided into multiple coding blocks or "generations" where coding is performed over each individual generation to lower the computational complexity of the encoding and decoding operations. Alternatively, a convolutional approach can be used, where coding is applied to overlapping spans of data symbols, possibly of different spanning widths, viewed as a sliding coding window. In generation-based RLNC, not all symbols within a single generation need to be present for coding to start. Similarly, a sliding window can be variable-sized, with more data symbols added to the coding window as they arrive. Thus, innovative coded symbols can be generated as data symbols arrive. This "on-the-fly" coding technique reduces coding delays at transmit buffers, and together with rateless encoding operations, enables the sender to start emitting coded packets as soon as data is received from an upper layer in the protocol stack, adapting to fluctuating incoming traffic flows. Injecting coded symbols based on a dynamic transmission window also breaks the decoding delay lower bound imposed by traditional block codes and is well suited for delay-sensitive applications and streaming protocols.

When coded symbols are transmitted through a communication network, erasures may occur, depending on channel conditions and interactions with underlying transport protocols. RLNC can efficiently repair such erasures, potentially improving protocol response to erasure events to ensure reliability and throughput over the communication network. For example, in a point-to-point connection, RLNC can proactively compensate for packet erasures by generating Forward Erasure Correcting (FEC) redundancy, especially when a packet erasure probability can be estimated. As any number of coded symbols may be

generated from a set of data symbols, RLNC is naturally suited for adapting to network conditions by adjusting redundancy dynamically to fit the level of erasures, and by updating coding parameters during a session. Alternatively, packet erasures may be repaired reactively by using feedback requests from the receiver to the sender, or by a combination of FEC and retransmission. RLNC simplifies state and feedback management and coordination as only a desired number of degrees of freedom needs to be communicated from the receiver to the sender, instead of indications of the exact packets to be retransmitted. The need to exchange packet arrival state information is therefore greatly reduced in feedback operations.

The advantages of RLNC in state and feedback management are apparent in a multicast setting. In this one-to-many setup, uncorrelated losses may occur, and any retransmitted data symbol is likely to benefit only a single receiver. By comparison, a transmitted RLNC coded symbol is likely to carry a new degree of freedom that may correct different errors at different receivers simultaneously. Similarly, RLNC offers advantages in coordinating multiple paths, multiple sources, mesh networking and cooperation, and peer-to-peer operations.

A more detailed introduction to network coding including RLNC is provided in the books [MS11] and [HL08].

## 1.2. Generation-Based RLNC

This section describes a generation-based RLNC scheme.

In generation-based RLNC, input data as received from an upper layer in the protocol stack is segmented into equal-sized blocks, denoted as generations, and each generation is further segmented into equal-sized data symbols for encoding, with paddings added when necessary. Encoding and decoding are performed over each individual generation. Figure 1 below provides an illustrative example where each generation includes four data symbols, and a systematic RLNC code is generated with rate  $2/3$ .

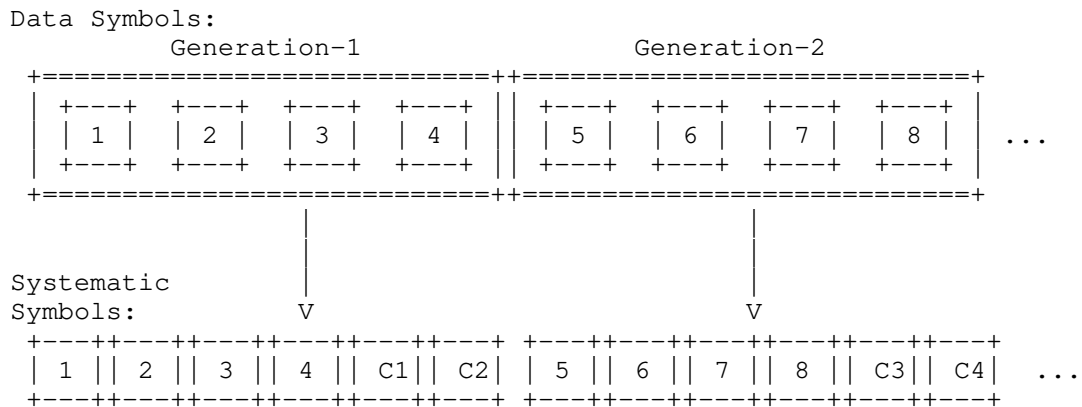


Figure 1: Generation-based RLNC with rate 2/3, systematic encoding performed on data symbols within each generation.

Symbols can be of any size, although symbol sizes typically depend on application or system specifications. In scenarios with highly varying input data frame sizes, a small symbol size may be desirable for achieving flexibility and transmission efficiency, with one or more symbols concatenated to form a coded data packet. In this context, existing basic FEC schemes [RFC5445] do not support the use of a single header for multiple coded symbols, whereas the symbol representation design as described in [Symbol-Representation] provides this option.

For any protocol that utilizes generation-based RLNC, a setup process is necessary for establishing a connection and conveying coding parameters from the sender to the receiver. Such coding parameters can include one or more of field size, code specifications, index of the current generation being encoded at the sender, generation size, code rate, and desired feedback frequency or probability. Some coding parameters are updated dynamically during the transmission process, reflecting the coding operations over sequences of generations, and adjusting to channel conditions and resource availability. For example, an outer header can be added to the symbol representation specified in [Symbol-Representation] to indicate the current generation encoded within the symbol representation. Such information is essential for proper recoding and decoding operations, but the exact design of the outer header is outside the scope of the current document. At the minimum, an outer header should indicate the current generation, generation size, symbol size, and field size. Section 2 provides a detailed discussion of coding parameter considerations.

### 1.3. Sliding Window RLNC

This section describes a sliding-window RLNC scheme. Sliding window RLNC was first described in [SS09].

In sliding-window RLNC, input data as received from an upper layer in the protocol stack is segmented into equal-sized data symbols for encoding. In some implementations, the sliding encoding window can expand in size as new data packets arrive, until it is closed off by an explicit instruction, such as a feedback message that re-initiates the encoding window. In some implementations, the size of the sliding encoding window is upper bounded by some parameter, fixed or dynamically determined by online behavior such as packet loss or congestion estimation. Figure 3 below provides an example of a systematic finite sliding window code with rate 2/3.

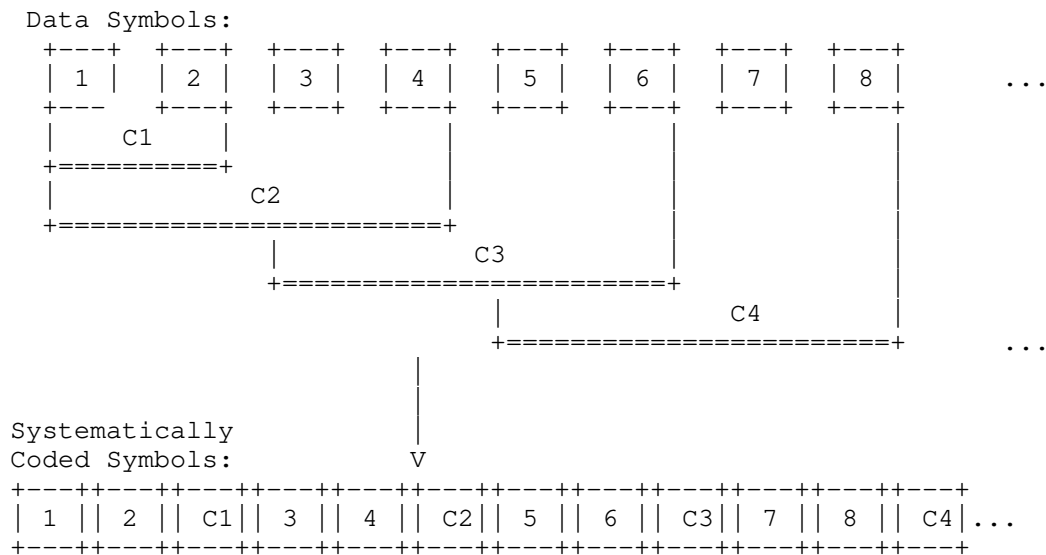


Figure 3: Finite sliding-window RLNC with code rate 2/3, systematic encoding performed on data symbols within the sliding coding window.

For any protocol that utilizes sliding-window RLNC, a setup process is necessary for establishing a connection and conveying coding parameters from the sender to the receiver. Such coding parameters can include one or more of field size, code specifications, symbol ordering, encoding window position, encoding window size, code rate, and desired feedback frequency or probability. Some coding parameters can also be updated dynamically during the transmission process in accordance to channel conditions and resource

availability. For example, an outer header can be added to the symbol representation specified in [Symbol-Representation] to indicate an encoding window position, as a starting index for current data symbols being encoded within the symbol representation. Again, such information is essential for proper recoding and decoding operations, but the exact design of the outer header is outside the scope of the current document. At the minimum, an outer header should indicate the current encoding window position, encoding window size, symbol size, and field size. Section 2 provides a detailed discussion of coding parameter considerations.

Once a connection is established, RLNC coded packets comprising one or more coded symbols are transmitted from the sender to the receiver. The sender can transmit in either a systematic or coded fashion, with or without receiver feedback. In progressive decoding of RLNC coded symbols, the notion of "seen" packets can be utilized to provide degree of freedom feedbacks. Seen packets are those packet that have contributed to a received coded packet, where generally the oldest such packet that has yet to be declared seen is declared as seen [SS09].

#### 1.4. Recoding

Recoding is the process where coded or partially decoded symbols are re-encoded without first being fully decoded. To recode, both coded symbols and corresponding coding coefficient vectors are linearly combined, respectively, with new randomly generated recoding coefficients. Recoded symbols and recoded coefficient vectors generally have the same size as before and are indistinguishable from the original coded symbols and coding coefficient vectors. Recoding is typically performed at intermediate network nodes, in either an intra-session or an inter-session fashion. Intra-session coding refers to coding between packets of the same flow or session, while inter-session coding refers to combination of packets from different flows or sessions in a network.

As recoding requires the same operations on the coding coefficient vectors as applied to the coded symbols, coding coefficients must be updated by recoding coefficients. This is generally achieved by having the coding coefficients accessible at recoding nodes so that they may be updated. Thus, either the original coding coefficients or reversible representations of the coding coefficients need to be communicated from upstream nodes to the recoding nodes.

## 2. Practical Considerations

This is an open section describing various practical considerations such as standardization approaches and implementation-related topics.

### 2.1. Symbol Representation

This sub-section argues for the specification of symbol representation as a starting point for network coding standardization and provides relevant coding parameter design considerations.

#### 2.1.1. Symbol Representation as a Standardization Approach

Symbol representation specifies the format of the symbol-carrying data unit that is to be coded, recoded, and decoded. In other words, symbol representation defines the format of the coding-layer data unit, including header format and symbol concatenation.

Network Coding has fundamentally different requirements from conventional point-to-point codes. Network coding owes its distinct requirements to its dynamic structure, leading to a highly reconfigurable symbol set. For example:

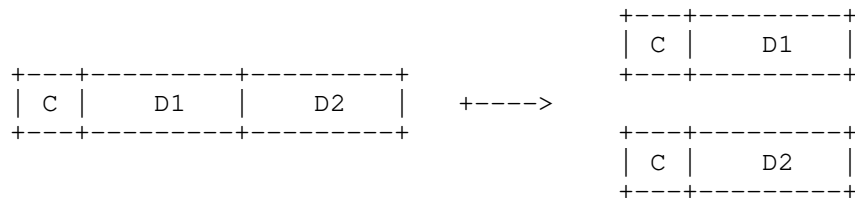
- o Coefficient Location: RLNC's encoding, recoding, and decoding process requires coefficients and payload to go through identical coding operations. These operations are independent from the location of the coefficients. As a consequence, coefficient location is flexible. While some designs cluster coefficients together, other designs may distribute them throughout the payload in a manner that is specific to a given protocol. [SS09]
- o Number of coefficients: RLNC is designed to allow coding and recoding even when the number of input symbols is dynamic, leading to varying code density. As a consequence, the number of coefficients and source data symbols need not be fixed.
- o Payload Size: Although an identical size of symbols is desirable when performing coding operations, padding and fragmentation are viable not only at the source but also throughout the network, as illustrated in the example of Figure 5. This allows flexibility in the payload size.
- o Field: Although the finite field is typically a fixed system variable, this is not necessarily the case. Network coding need not specify a single field for all payload components, as different symbols may belong to different fields (e.g., packet concatenation). This feature does not necessarily complicate

coding, since finite field operations defined in a given field are typically valid in multiple other fields.

Useful symbol representations should include provisions for the major coding functions that are relevant to the application, such as recoding, feedback, or inter-session network coding. For example, recoding requires the coefficients to be accessible at the intermediate recoding nodes. Hence, architectures and protocols requiring recoding must specify coefficient location.

Furthermore, the example of Figure 4 illustrates how the knowledge of coefficient location affects the way a coded payload is fragmented, coded, and transported across the network.

(a) Code-aware fragmentation



(b) Conventional fragmentation

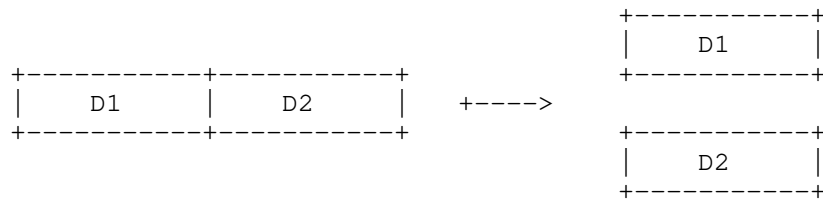


Figure 4: Network operations such as fragmentation may be affected by symbol representation. For example, whether coefficient location is (a) specified or (b) hidden may affect the way fragmentation is carried out.

In Figure 4 (a), coefficient locations are known, allowing the association of the coefficient set C to both fragments D1 and D2 of original payload. The ability to manipulate the original coefficients allows the newly formed packets to be recoded and decoded at the same coding layer. Figure 4 (b) shows a case where the coding coefficient location are unknown. This may occur when the file is pre-coded by a higher layer such as the application layer, or when coefficients are deliberately hidden for security reasons, leading to typical fragmentation without coefficient replication.

The absence of information on coefficient location has important implications. One such implication is that any additional coding needs to be carried out within a new coding layer, potentially leading to higher computational and transport overheads.

The elements discussed above demonstrate that the design choices related to symbol representation have a direct impact on the viability of protocols, topologies, and architecture. The importance of symbol representation is illustrated in Figure 5, where the term "architecture" includes coding architecture (e.g., generation or sliding window), the layer placement of coding operations, and coding objectives (e.g., erasure correction, multisourcing, etc.).

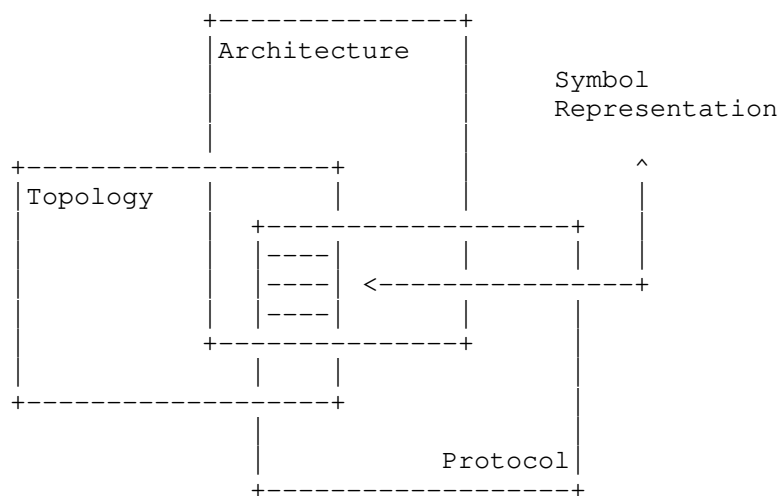


Figure 5: The specification of symbol representation has major implications on system architecture, topology, and protocol.

Since symbol representation has implications on core design elements, it is expected that coding implementations that share protocol, architecture, and topology elements are likely to reuse the same symbol representation. For example, implementations with security requirements can reuse a common symbol representation that hides coefficient locations.

Another example can be found in [Symbol-Representation], which specifies symbol representation designs for generation-based and sliding window RLNC implementations. These designs introduce highly reusable formats that concatenate multiple symbols and associate them with a single symbol representation header.



### 2.1.1.2. Coding Parameter Design Considerations

For any protocol that utilizes generation-based or sliding-window RLNC, several coding parameters must be communicated from the sender to the receiver as part of the protocol design. Without elaborating on all such parameters, this section examines those essential for symbol representation design, including field size, symbol size, maximum number of symbols, and maximum generation or window size.

As RLNC is performed over a finite field, field size determines the complexity of the required mathematical computations. Larger field sizes translate to higher probability of successful decoding, as randomly generated coding coefficient vectors are more likely to be independent from each other. However, larger field sizes may also result in higher computational complexity, leading to longer decoding latency, higher energy usage, and other hardware requirements for both the encoder and the decoder. A finite field size of 2 or the binary Finite Field  $FF(2)$  should always be supported since addition, multiplication, and division over  $FF(2)$  are equivalent to elementary XOR, AND, and IDENTITY operations respectively. It is also desirable to support a field size of  $2^8$ , corresponding to a single byte, and where operations are performed over the binary extension field  $FF(2^8)$  with polynomial  $x^8+x^4+x^3+x^2+1$ .

The choice of symbol size typically depends on the application or system specification. For example, a symbol size may be chosen based on the size of a maximum transmission unit (MTU) so that datagrams are not fragmented as they traverse a network, while also ensuring no symbol bits are unnecessarily wasted. A symbol representation design can be flexible and accommodate any symbol size in bytes. For example, an IP packet is typically in the range between 500 and 1500 bytes, while a much smaller datagram having a size of 90 bytes may be used by satellite communication networks. The symbol representation in [Symbol-Representation] can be configured to support either or both cases in different implementations.

The generation size or coding window size is a tradeoff between the strength of the code and the computational complexity of performing the coding operations. With a larger generation/window size, fewer generations or coding windows are needed to enclose a data message of a given size, thus reducing protocol overhead for coordinating individual generations or coding windows. In addition, a larger generation/window size increases the likelihood that a received coded symbol is innovative with respect to previously received symbols, thus amortizing retransmission or FEC overheads. Conversely, when coding coefficients are attached, larger generation/window sizes also lead to larger overheads per packet. The generation/window size to

be used can be signaled between the sender and receiver when the connection is first established.

Lastly, to successfully decode RLNC coded symbols, sufficient degrees of freedom are required at the decoder. The maximum number of redundant symbols that can be transmitted is therefore limited by the number of linearly independent coding coefficient vectors that can be supported by the system. For example, if coding vectors are constructed using a pseudo-random generator, the maximum number of redundant symbols that can be transmitted is limited by the number of available generator states.[RFC5445]

### 3. Security Considerations

This document does not present new security considerations.

### 4. IANA Considerations

This document has no actions for IANA.

### 5. References

#### 5.1. 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>>.
- [RFC3453] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, DOI 10.17487/RFC3453, December 2002, <<https://www.rfc-editor.org/info/rfc3453>>.
- [RFC5445] Watson, M., "Basic Forward Error Correction (FEC) Schemes", RFC 5445, DOI 10.17487/RFC5445, March 2009, <<https://www.rfc-editor.org/info/rfc5445>>.
- [Symbol-Representation] Heide, J., Shi, S., Fouli, K., Medard, M., and V. Chook, "Random Linear Network Coding (RLNC)-Based Symbol Representation", February 2018, <<https://www.ietf.org/archive/id/draft-heide-nwcrg-rlnc-00.txt>>.

## 5.2. Informative References

- [HK03] Ho, T., Koetter, R., Medard, M., Karger, D., and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting", July 2003, <<http://ieeexplore.ieee.org/document/1228459/>>.
- [HL08] Ho, T. and D. Lun, "Network Coding: An Introduction", April 2008.
- [MS11] Medard, M. and A. Sprintson, "Network Coding: Fundamentals and Applications", October 2011.
- [SS09] Sundararajan, J., Shah, D., Medard, M., Mitzenmacher, M., and J. Barros, "Network Coding Meets TCP", April 2009, <<http://ieeexplore.ieee.org/document/5061931/>>.

## Authors' Addresses

Janus Heide  
Steinwurf Aps  
Aalborg  
Denmark

Email: [janus@steinwurf.com](mailto:janus@steinwurf.com)

Shirley Shi  
Code On Network Coding LLC  
Cambridge  
USA

Email: [xshi@alum.mit.edu](mailto:xshi@alum.mit.edu)

Kerim Fouli  
Code On Network Coding LLC  
Cambridge  
USA

Email: [fouli@codeontechnologies.com](mailto:fouli@codeontechnologies.com)

Muriel Medard  
Code On Network Coding LLC  
Cambridge  
USA

Email: [muriel.medard@codeontechnologies.com](mailto:muriel.medard@codeontechnologies.com)

Vince Chook  
Inmarsat PLC  
London  
United Kingdom

Email: [Vince.Chook@inmarsat.com](mailto:Vince.Chook@inmarsat.com)

NetWork Communications Research Group (NWCRG)  
Internet-Draft  
Intended status: Informational  
Expires: May 3, 2021

N. Kuhn, Ed.  
CNES  
E. Lochin, Ed.  
ENAC  
October 30, 2020

Network coding for satellite systems  
draft-irtf-nwcrg-network-coding-satellites-15

Abstract

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

The objective is to contribute to a larger deployment of network coding techniques in and above the network layer in satellite communication systems. The document also identifies open research issues related to the deployment of network coding in satellite communication systems.

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 May 3, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. A Note on Satellite Networks Topology . . . . .	3
3. Use-cases for Improving SATCOM System Performance Using Network Coding . . . . .	5
3.1. Two-way Relay Channel Mode . . . . .	5
3.2. Reliable Multicast . . . . .	5
3.3. Hybrid Access . . . . .	6
3.4. LAN Packet Losses . . . . .	7
3.5. Varying Channel Conditions . . . . .	8
3.6. Improving Gateway Handover . . . . .	8
4. Research Challenges . . . . .	9
4.1. Joint-use of Network Coding and Congestion Control in SATCOM Systems . . . . .	9
4.2. Efficient Use of Satellite Resources . . . . .	10
4.3. Interaction with Virtualized Satellite Gateways and Terminals . . . . .	10
4.4. Delay/Disruption Tolerant Networking (DTN) . . . . .	10
5. Conclusion . . . . .	11
6. Glossary . . . . .	11
7. Acknowledgements . . . . .	13
8. IANA Considerations . . . . .	13
9. Security Considerations . . . . .	13
10. Informative References . . . . .	13
Authors' Addresses . . . . .	16

## 1. Introduction

This document is one product of and represents the collaborative work and consensus of the Coding for Efficient Network Communications Research Group (NWCRCG); while it is not an IETF product and not a standard it intends to inform the SATellite COMmunication (SATCOM) and Internet research communities about recent developments in Network Coding. A glossary is included in Section 6 to clarify the terminology use throughout the document.

As will be shown in this document, the implementation of network coding techniques above the network layer, at application or transport layers (as described in [RFC1122]), offers an opportunity for improving the end-to-end performance of SATCOM systems. While physical- and link-layer coding error protection is usually enough to

provide Quasi-Error Free transmission thus minimizing packet loss, when residual errors at those layers cause packet losses, retransmissions add significant delays (in particular in geostationary systems with over 0.7 second round-trip delays). Hence the use of network coding at the upper layers can improve the quality of service in SATCOM subnetworks and eventually favorably impact the experience of end users.

While there is an active research community working on network coding techniques above the network layer in general and in SATCOM in particular, not much of this work has been deployed in commercial systems. In this context, this document identifies opportunities for further usage of network coding in commercial SATCOM networks.

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

- o Channel and link error correcting codes are considered part of the PHYsical (PHY) layer error protection and are out of the scope of this document.
- o Forward Erasure Correction (FEC) (also called Application-Level FEC) operates above the link layer and targets packet loss recovery.
- o This document considers only coding (or coding techniques or coding schemes) that use a linear combination of packets and excludes for example content coding (e.g., to compress a video flow) or other non-linear operation.

## 2. A Note on Satellite Networks Topology

There are multiple SATCOM systems, for example broadcast TV, point to point communication or IoT monitoring. Therefore, depending on the purpose of the system, the associated ground segment architecture will be different. This section focuses on a satellite system that follows the European Telecommunications Standards Institute (ETSI) Digital Video Broadcasting (DVB) standards to provide broadband Internet access via ground-based gateways [ETSIEN2014]. One must note that the overall data capacity of one satellite may be higher than the capacity that one single gateway supports. Hence, there are usually multiple gateways for one unique satellite platform.

In this context, Figure 1 shows an example of a multi-gateway satellite system, where BBFRAME stands for Base-Band FRAME, PLFRAME for Physical Layer FRAME and PEP for Performance Enhancing Proxy. More information on a generic SATCOM ground segment architecture for bidirectional Internet access can be found in [SAT2017].

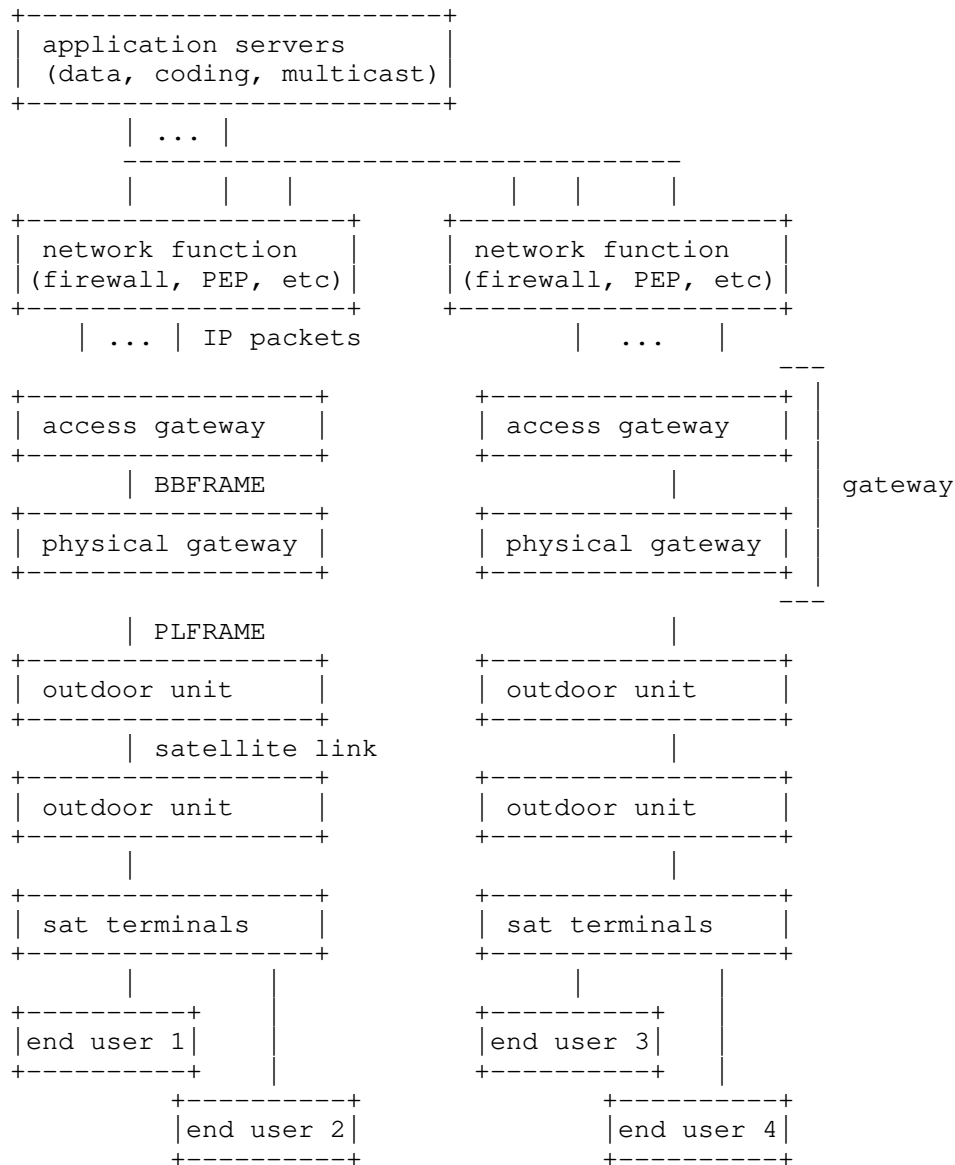


Figure 1: Data plane functions in a generic satellite multi-gateway system. More details can be found in DVB standard documents.



### 3. Use-cases for Improving SATCOM System Performance Using Network Coding

This section details use-cases where network coding techniques could improve SATCOM system performance.

#### 3.1. Two-way Relay Channel Mode

This use-case considers two-way communication between end-users, through a satellite link as seen in Figure 2.

Satellite terminal A sends a packet flow A and satellite terminal B sends a packet flow B to a coding server. The coding server then sends a combination of both flows instead of each individual flows. This results in non-negligible capacity savings that has been demonstrated in the past [ASMS2010]. In the example, a dedicated coding server is introduced (note that its location could be different based on deployment use-case). The network coding operations could also be done at the satellite level, although this would require a lot of computational resources on-board and may not be supported by today's satellites.

-X}- : traffic from satellite terminal X to the server  
 ={X+Y}= : traffic from X and Y combined sent from  
           the server to terminals X and Y

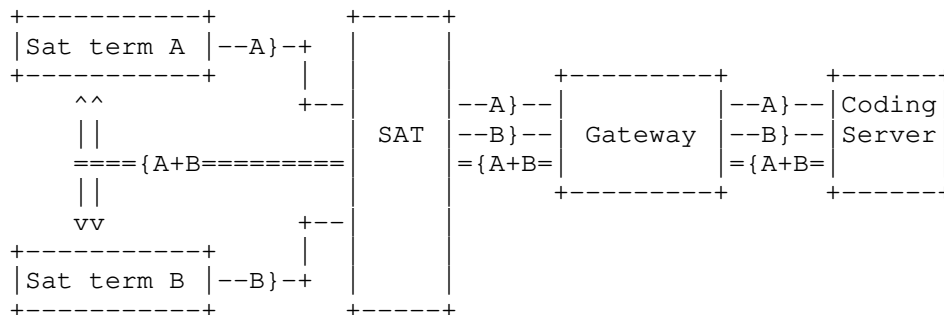


Figure 2: Network Architecture for Two-way Relay Channel using NC

#### 3.2. Reliable Multicast

The use of multicast servers is one way to better utilize satellite broadcast capabilities. As one example satellite-based multicast is proposed in the SHINE ESA project [I-D.vazquez-nfvrg-netcod-function-virtualization] [SHINE]. This use-case considers adding redundancy to a multicast flow depending on what has been received by different end-users, resulting in non-



Depending on the protocol, network coding could be applied at each of the Customer Premises Equipment (CPE) and at the concentrator or both. Apart from packet losses, other gains from this approach include a better tolerance to out-of-order packet delivery which occur when exploited links exhibit high asymmetry in terms of Round-Trip Time (RTT). Depending on the ground architecture [I-D.chin-nfvrg-cloud-5g-core-structure-yang] [SAT2017], some ground equipment might be hosting both SATCOM and cellular network functionality.

-{}- : bidirectional link

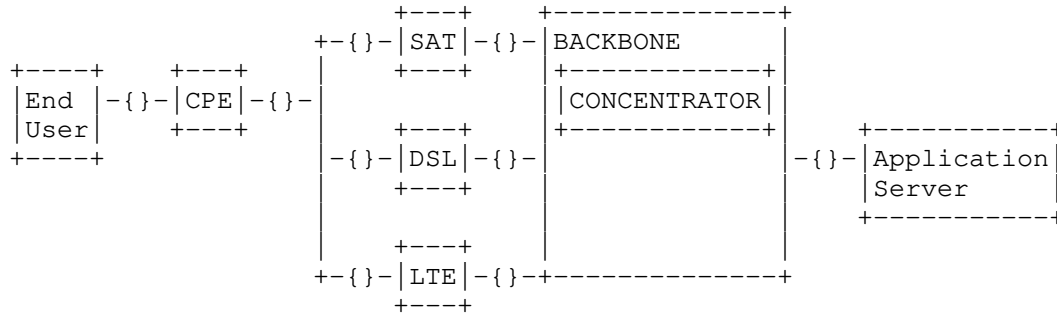


Figure 4: Network Architecture for a Hybrid Access Using Network Coding

### 3.4. LAN Packet Losses

This use-case considers using network coding in the scenario where a lossy WIFI link is used to connect to the SATCOM network. When encrypted end-to-end applications based on UDP are used, a Performance Enhancing Proxy (PEP) cannot operate hence other mechanism need to be used. The WIFI packet losses will result in an end-to-end retransmission that will harm the end-user quality of experience and poorly utilize SATCOM bottleneck resource for non-revenue generating traffic. In this use-case, adding network coding techniques will prevent the end-to-end retransmission from occurring since the packet losses would probably be recovered.

The architecture is shown in Figure 5.

-{}- : bidirectional link  
 -''- : Wi-Fi link  
 C : where network coding techniques could be introduced

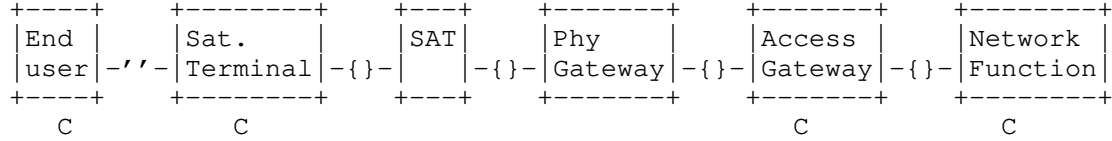


Figure 5: Network Architecture for dealing with LAN Losses

### 3.5. Varying Channel Conditions

This use-case considers the usage of network coding to cope with sub second physical channel condition changes where the physical-layer mechanisms (Adaptive Coding and Modulation (ACM)) may not adapt the modulation and error-correction coding in time: the residual errors lead to higher layer packet losses that can be recovered with network coding. This use-case is mostly relevant when mobile users are considered or when the satellite frequency band introduces quick changes in channel condition (Q/V bands, Ka band, etc.). Depending on the use-case (e.g., very high frequency bands, mobile users), the relevance of adding network coding is different.

The system architecture is shown in Figure 6.

-{}- : bidirectional link  
 C : where network coding techniques could be introduced

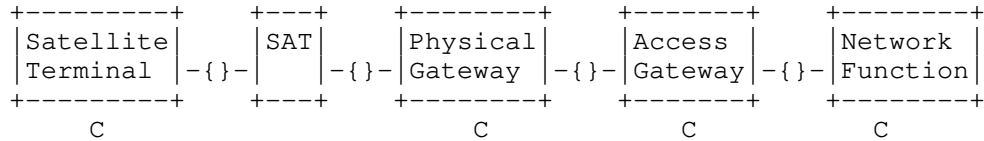


Figure 6: Network Architecture for dealing with Varying Link Characteristics

### 3.6. Improving Gateway Handover

This use-case considers the recovery of packets that may be lost during gateway handover. Whether for off-loading a given equipment or because the transmission quality differs from gateway to gateway, switching the transmission gateway may be beneficial. However, packet losses can occur if the gateways are not properly synchronized or if the algorithm used to trigger gateway handover is not properly tuned. During these critical phases, network coding can be added to

improve the reliability of the transmission and allow a seamless gateway handover.

Figure 7 illustrates this use-case.

-{}- : bidirectional link

! : management interface

C : where network coding techniques could be introduced

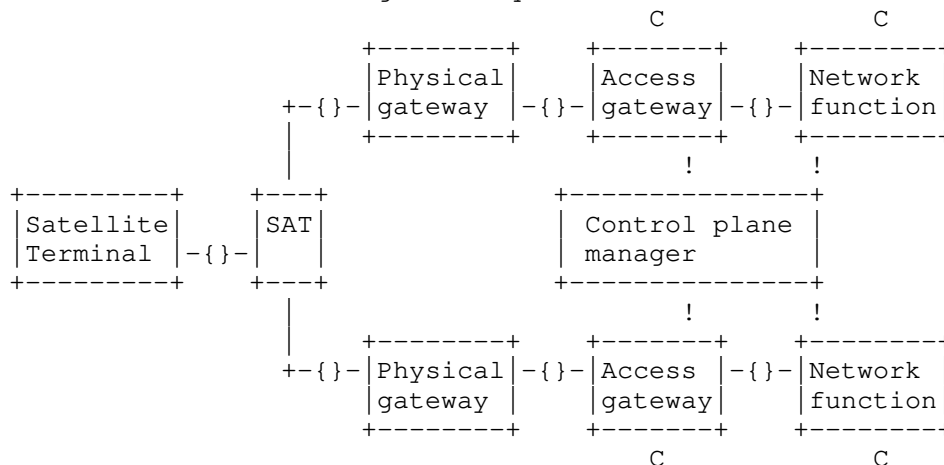


Figure 7: Network Architecture for dealing with Gateway Handover

#### 4. Research Challenges

This section proposes a few potential approaches to introduce and use network coding in SATCOM systems.

##### 4.1. Joint-use of Network Coding and Congestion Control in SATCOM Systems

Many SATCOM systems typically use Performance Enhancing Proxy (PEP) RFC 3135 [RFC3135]. PEPs usually split end-to-end connections and forward transport or application layer packets to the satellite baseband gateway. PEPs contribute to mitigate congestion in a SATCOM systems by limiting the impact of long delays on Internet protocols. A PEP mechanism could also include network coding operation and thus support the use-cases that have been discussed in the Section 3 of this document.

Deploying network coding in the PEP could be relevant and be independent from the specifics of a SATCOM link. This however leads to research questions dealing with the potential interaction between

network coding and congestion control. This is discussed in [I-D.irtf-nwcrg-coding-and-congestion].

#### 4.2. Efficient Use of Satellite Resources

There is a recurrent trade-off in SATCOM systems: how much overhead from redundant reliability packets can be introduced to guarantee a better end-user QoE while optimizing capacity usage? At which layer this supplementary redundancy should be added?

This problem has been tackled in the past by the deployment of physical-layer error-correction codes, but there remains questions on adapting the coding overhead and added delay for, e.g., the quickly varying channel conditions use-case where ACM may not be reacting quickly enough as was discussed in Section 3.5. The higher layer with network coding does not react more quickly than the physical layer, but may operate over a packet-based time window that is larger than the physical one.

#### 4.3. Interaction with Virtualized Satellite Gateways and Terminals

In the emerging virtualized network infrastructure, network coding could be easily deployed as Virtual Network Functions (VNF). The next generation of SATCOM ground segments will rely on a virtualized environment to integrate to terrestrial networks. This trend towards Network Function Virtualization (NFV) is also central to 5G and next generation cellular networks, making this research applicable to other deployment scenarios [I-D.chin-nfvrg-cloud-5g-core-structure-yang]. As one example, the network coding VNF deployment in a virtualized environment has been presented in [I-D.vazquez-nfvrg-netcod-function-virtualization].

A research challenge would be the optimization of the NFV service function chaining, considering a virtualized infrastructure and other SATCOM specific functions, in order to guarantee efficient radio-link usage and provide easy-to-deploy SATCOM services. Moreover, another challenge related to a virtualized SATCOM equipment is the management of limited buffered capacities in large gateways.

#### 4.4. Delay/Disruption Tolerant Networking (DTN)

Communications among deep-space platforms and terrestrial gateways can be a challenge. Reliable end-to-end (E2E) communications over such paths must cope with very long delays and frequent link disruptions; indeed, E2E connectivity may only be available intermittently, if at all. Delay/Disruption Tolerant Networking (DTN) [RFC4838] is a solution to enable reliable internetworking space communications where both standard ad-hoc routing and E2E

Internet protocols cannot be used. Moreover, DTN can also be seen as an alternative solution to transfer data between a central PEP and a remote PEP.

Network Coding enables E2E reliable communications over a DTN with potential adaptive re-encoding, as proposed in [THAI15]. Here, the use-cases proposed in Section 3.5 would encourage the usage of network coding within the DTN stack to improve the physical channel utilization and minimize the effects of the E2E transmission delays. In this context, the use of packet erasure coding techniques inside a Consultative Committee for Space Data Systems (CCSDS) architecture has been specified in [CCSDS-131.5-0-1]. One research challenge remains on how such network coding can be integrated in the IETF DTN stack.

## 5. Conclusion

This document introduces some wide-scale network coding technique opportunities in satellite telecommunications systems.

Even though this document focuses on satellite systems, it is worth pointing out that some scenarios proposed here may be relevant to other wireless telecommunication systems. As one example, the generic architecture proposed in Figure 1 may be mapped onto cellular networks as follows: the 'network function' block gathers some of the functions of the Evolved Packet Core subsystem, while the 'access gateway' and 'physical gateway' blocks gather the same type of functions as the Universal Mobile Terrestrial Radio Access Network. This mapping extends the opportunities identified in this document since they may also be relevant for cellular networks.

## 6. Glossary

The glossary of this memo extends the glossary of the taxonomy document [RFC8406] as follows:

- o ACM : Adaptive Coding and Modulation;
- o BBFRAME: Base-Band FRAME - satellite communication layer 2 encapsulation work as follows: (1) each layer 3 packet is encapsulated with a Generic Stream Encapsulation (GSE) mechanism, (2) GSE packets are gathered to create BBFRAMEs, (3) BBFRAMEs contain information related to how they have to be modulated (4) BBFRAMEs are forwarded to the physical-layer;
- o CPE: Customer Premises Equipment;
- o COM: COMMunication;

- o DSL: Digital Subscriber Line;
- o DTN: Delay/Disruption Tolerant Networking;
- o DVB: Digital Video Broadcasting;
- o E2E: End-to-end;
- o ETSI: European Telecommunications Standards Institute;
- o FEC: Forward Erasure Correction;
- o FLUTE: File Delivery over Unidirectional Transport [RFC6726];
- o IntraF: Intra-Flow Coding;
- o InterF: Inter-Flow Coding;
- o IoT: Internet of Things;
- o LTE: Long Term Evolution;
- o MPC: Multi-Path Coding;
- o NC: Network Coding;
- o NFV: Network Function Virtualization - concept of running software-defined network functions;
- o NORM: NACK-Oriented Reliable Multicast [RFC5740];
- o PEP: Performance Enhancing Proxy [RFC3135] - a typical PEP for satellite communications include compression, caching and TCP ACK spoofing and specific congestion control tuning;
- o PLFRAME: Physical Layer FRAME - modulated version of a BBFRAME with additional information (e.g., related to synchronization);
- o QEF: Quasi-Error-Free;
- o QoE: Quality-of-Experience;
- o QoS: Quality-of-Service;
- o RTT: Round-Trip Time;
- o SAT: SATellite;



- o SATCOM: generic term related to all kinds of SATellite COMMunication systems;
- o SPC: Single-Path Coding;
- o VNF: Virtual Network Function - implementation of a network function using software.

## 7. Acknowledgements

Many thanks to John Border, Stuart Card, Tomaso de Cola, Vincent Roca, Lloyd Wood and Marie-Jose Montpetit for their help in writing this document.

## 8. IANA Considerations

This memo includes no request to IANA.

## 9. Security Considerations

Security considerations are inherent to any access network, and in particular SATCOM systems. Such as it is done in cellular networks, over-the-air data can be encrypted using e.g. [ETSITS2011]. Because the operator may not enable this [SSP-2020], the applications should apply cryptographic protection. The use of FEC or Network Coding in SATCOM comes with risks (e.g., a single corrupted redundant packet may propagate to several flows when they are protected together in an Inter-Flow coding approach, see section Section 3). While this document does not further elaborate on this, the security considerations discussed in [RFC6363] apply.

## 10. Informative References

[ASMS2010]

De Cola, T. and et. al., "Demonstration at opening session of ASMS 2010", Advanced Satellite Multimedia Systems (ASMS) Conference , 2010.

[CCSDS-131.5-0-1]

"Erasure correcting codes for use in near-earth and deep-space communications", CCSDS Experimental specification 131.5-0-1, 2014.

[ETSIEN2014]

"Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard", ETSI EN 301 545-2, 2014.

- [ETSI TR2017]  
"Satellite Earth Stations and Systems (SES); Multi-link routing scheme in hybrid access network with heterogeneous links", ETSI TR 103 351, 2017.
- [ETSI TS2011]  
"Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM); Part 5: CPCM Security Toolbox", ETSI TS 102 825-5, 2011.
- [I-D.chin-nfvrg-cloud-5g-core-structure-yang]  
Chen, C. and Z. Pan, "Yang Data Model for Cloud Native 5G Core structure", draft-chin-nfvrg-cloud-5g-core-structure-yang-00 (work in progress), December 2017.
- [I-D.irtf-nwcr-g-coding-and-congestion]  
Kuhn, N., Lochin, E., Michel, F., and M. Welzl, "Coding and congestion control in transport", draft-irtf-nwcr-g-coding-and-congestion-03 (work in progress), July 2020.
- [I-D.vazquez-nfvrg-netcod-function-virtualization]  
Vazquez-Castro, M., Do-Duy, T., Romano, S., and A. Tulino, "Network Coding Function Virtualization", draft-vazquez-nfvrg-netcod-function-virtualization-02 (work in progress), November 2017.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, DOI 10.17487/RFC3135, June 2001, <<https://www.rfc-editor.org/info/rfc3135>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [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>>.
- [RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <<https://www.rfc-editor.org/info/rfc6726>>.
- [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>>.
- [RFC8681] Roca, V. and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME", RFC 8681, DOI 10.17487/RFC8681, January 2020, <<https://www.rfc-editor.org/info/rfc8681>>.
- [SAT2017] Ahmed, T., Dubois, E., Dupe, JB., Ferrus, R., Gelard, P., and N. Kuhn, "Software-defined satellite cloud RAN", International Journal on Satellite Communications and Networking vol. 36 - <https://doi.org/10.1002/sat.1206>, 2017.
- [SHINE] Pietro Romano, S. and et. al., "Secure Hybrid In Network caching Environment (SHINE) ESA project", ESA project , 2017 on-going.
- [SSP-2020] Pavur (et al.), J., "A Tale of Sea and SkyOn the Security of Maritime VSAT Communications", IEEE Symposium on Security and Privacy 10.1109/SP40000.2020.00056, 2020.
- [THAI15] Thai, T., Chaganti, V., Lochin, E., Lacan, J., Dubois, E., and P. Gelard, "Enabling E2E reliable communications with adaptive re-encoding over delay tolerant networks", Proceedings of the IEEE International Conference on Communications <http://dx.doi.org/10.1109/ICC.2015.7248441>, June 2015.

## Authors' Addresses

Nicolas Kuhn (editor)  
CNES  
18 avenue Edouard Belin  
Toulouse 31400  
France

Email: nicolas.kuhn@cnes.fr

Emmanuel Lochin (editor)  
ENAC  
7 avenue Edouard Belin  
Toulouse 31400  
France

Email: emmanuel.lochin@enac.fr

NWCRG and ICNRG  
Internet-Draft  
Intended status: Informational  
Expires: August 31, 2022

K. Matsuzono  
H. Asaeda  
NICT  
C. Westphal  
Huawei  
February 27, 2022

Network Coding for Content-Centric Networking / Named Data Networking:  
Considerations and Challenges  
draft-irtf-nwcrg-nwc-ccn-reqs-09

Abstract

This document describes the current research outcomes in Network Coding (NC) for Content-Centric Networking (CCNx) / Named Data Networking (NDN), and clarifies the technical considerations and potential challenges for applying NC in CCNx/NDN. This document is the product of the Coding for Efficient Network Communications Research Group (NWCRG) and the Information-Centric Networking Research Group (ICNRG).

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 August 31, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	4
2.1. Definitions related to NC . . . . .	4
2.2. Definitions related to CCNx/NDN . . . . .	6
3. CCNx/NDN Basics . . . . .	6
4. NC Basics . . . . .	7
5. Advantages of NC and CCNx/NDN . . . . .	8
6. Technical Considerations . . . . .	9
6.1. Content Naming . . . . .	9
6.1.1. Unique Naming for NC Packets . . . . .	9
6.1.2. Non-Unique Naming for NC Packets . . . . .	10
6.2. Transport . . . . .	11
6.2.1. Scope of NC . . . . .	11
6.2.2. Consumer Operation . . . . .	11
6.2.3. Forwarder Operation . . . . .	12
6.2.4. Producer Operation . . . . .	13
6.2.5. Backward Compatibility . . . . .	14
6.3. In-network Caching . . . . .	14
6.4. Seamless Consumer Mobility . . . . .	15
7. Challenges . . . . .	15
7.1. Adoption of Convolutional Coding . . . . .	15
7.2. Rate and Congestion Control . . . . .	16
7.3. Security . . . . .	16
7.4. Routing Scalability . . . . .	17
8. Security Considerations . . . . .	17
9. Acknowledgements . . . . .	18
10. Informative References . . . . .	19
Authors' Addresses . . . . .	22

## 1. Introduction

Information-Centric Networking (ICN) in general, and Content-Centric Networking (CCNx) [16] or Named Data Networking (NDN) [19] in particular, have emerged as a novel communication paradigm advocating the retrieval of data based on their names. This paradigm pushes content awareness into the network layer. It is expected to enable consumers to obtain the content they desire in a straightforward and efficient manner from the heterogenous networks they may be connected to. The CCNx/NDN architecture has introduced innovative ideas and has stimulated research in a variety of areas, such as in-network

caching, name-based routing, multipath transport, and content security. One key benefit of requesting content by name is that it eliminates the requirement to establish a session between the client and a specific server, and the content can thereby be retrieved from multiple sources.

In parallel, there has been a growing interest in both academia and industry for better understanding the fundamental aspects of Network Coding (NC) toward enhancing key system performance metrics such as data throughput, robustness and reduction in the required number of transmissions through connected networks, and redundant transmission on broadcast or point-to-multipoint connections. NC is a technique that is typically used for encoding packets to recover from lost source packets at the receiver, and for effectively obtaining the desired information in a fully distributed manner. In addition, in terms of security aspects, NC can be managed using a practical security scheme that deals with pollution attacks [2], and can be utilized for preventing eavesdroppers from obtaining meaningful information [3] or protecting a wireless video stream while retaining the NC benefits [4] [5].

From the perspective of the NC transport mechanism, NC can be divided into two major categories: coherent NC, and non-coherent NC [38] [39]. In coherent NC, the source and destination nodes know the exact network topology and the coding operations available at intermediate nodes. When multiple consumers are attempting to receive the same content such as live video streaming, coherent NC could enable optimal throughput by sending the content flow over the constructed optimal multicast trees [32]. However, it requires a fully adjustable and specific routing mechanism, and a large computational capacity for central coordination. In the case of non-coherent NC, which often uses Random Linear Coding (RLC), it is not necessary to know the network topology nor the intermediate coding operations [33]. As non-coherent NC works in a completely independent and decentralized manner, this approach is more feasible in terms of eliminating such a central coordination.

NC combines multiple packets together with parts of the same content, and may do this at the source and/or at other nodes in the network. Network coded packets are not associated with a specific server, as they may have been combined within the network. As NC is focused on what information should be encoded in a network packet instead of the specific host at which it has been generated, it is in line with the architecture of the CCNx/NDN core networking layer. NC allows for recovery of missing packets by encoding the information across several packets. ICN is synergistic with NC, as it allows for caching of data packets throughout the network. In a typical network that uses NC, the sender must transmit enough packets to retrieve the

original data. ICN offers an opportunity to retrieve network coded packets directly from caches in the network and this makes the combination of ICN and NC particularly effective. In fact, NC has already been implemented for information/content dissemination [6] [7] [8]. Montpetit, et al., first suggested that NC techniques be exploited to enhance key aspects of system performance in ICN [9]. Although CCNx/NDN excels to exploit the benefits of NC (as described in Section 5), some technical considerations are needed to combine NC and CCNx/NDN owing to the unique features of CCNx/NDN (as described in Section 6).

In this document, we consider how NC can be applied to the CCNx/NDN architecture and describe the technical considerations and potential challenges for making CCNx/NDN-based communications better using the NC technology. It should be noted that the presentation of specific solutions (e.g., NC optimization methods) for enhancing CCNx/NDN performance metrics by exploiting NC is outside the scope of this document.

This document represents the collaborative work and consensus of the Coding for Efficient Network Communications Research Group (NWCGRG) and the Information-Centric Networking Research Group (ICNRG). It is not an IETF product and is not a standard.

## 2. Terminology

### 2.1. Definitions related to NC

This section provides the terms related to NC used in this document, which are defined in RFC8406 [21] produced by NWCGRG.

- o Source Packet: A packet originating from the source that contributes to one or more source symbols. The source symbol is a unit of data originating from the source that is used as input to encoding operations. For instance, a real-time transport protocol (RTP) packet as a whole can constitute a source symbol. In other situations (e.g., to address variable size packets), a single RTP packet may contribute to various source symbols.
- o Repair Packet: A packet containing one or more coded symbols (also called repair symbol). Coded symbol or repair symbol is a unit of data that is the result of a coding operation, applied either to source symbols or (in case of re-coding) source and/or coded symbols. When there is a single repair symbol per repair packet, a repair symbol corresponds to a repair packet.
- o Encoding versus Re-coding versus Decoding: Encoding is an operation that takes source symbols as input and produces encoding



symbols (source or coded symbols) as output. Re-coding is an operation that takes encoding symbols as input and produces encoding symbols as output. Decoding is an operation that takes encoding symbols as input and produces source symbols as output.

The terms regarding coding types are defined as follows:

- o Random Linear Coding (RLC): A particular form of linear coding using a set of random coding coefficients. Linear coding performs linear combination of a set of input symbols (i.e., source and/or coded symbols) using a given set of coefficients and results in a repair symbol.
- o Block Coding: A coding technique wherein the input flow(s) must be first segmented into a sequence of blocks; encoding and decoding are performed independently on a per-block basis.
- o Sliding Window Coding or Convolutional Coding: A general class of coding techniques that rely on a sliding encoding window. Encoding window is a set of source (and coded in the case of re-coding) symbols used as input to the coding operations. The set of symbols change over time, as the encoding window slides over the input flow(s). This is an alternative solution to block coding.
- o Fixed or Elastic Sliding Window Coding: A coding technique that generates coded symbol(s) on the fly, from the set of source symbols present in the sliding encoding window at that time, usually by using linear coding. The sliding window may be either of fixed size or of variable size over the time (also known as "Elastic Sliding Window"). For instance, the size may depend on acknowledgments sent by the receiver(s) for a particular source symbol or source packet (received, decoded, or decodable).

The terms regarding low-level coding aspects are defined as follows:

- o Rank of the Linear System or Degrees of Freedom: At a receiver, the number of linearly independent equations of the linear system. It is also known as "Degrees of Freedom". The system may be of "full rank," wherein decoding is possible, or "partial rank", wherein only partial decoding is possible.
- o Generation, or Block: With block codes, the set of source symbols of the input flow(s) that are logically grouped into a block, before doing encoding.
- o Generation Size, or Block Size: With block codes, the number of source symbols belonging to a block. It is equivalent to the

number of source packets when there is a single source symbol per source packet.

- o Coding Coefficient: With linear coding, this is a coefficient in a certain finite field. This coefficient may be chosen in different ways: for instance, randomly, in a predefined table, or using a predefined algorithm plus a seed.
- o Coding Vector: A set of coding coefficients used to generate a certain coded symbol through linear coding.
- o Finite Field: Finite fields, used in linear codes, have the desired property of having all elements (except zero) invertible for + and \* and no operation over any elements can result in an overflow or underflow. Examples of finite fields are prime fields  $\{0..p^m-1\}$ , where  $p$  is prime. Most used fields use  $p=2$  and are called binary extension fields  $\{0..2^m-1\}$ , where  $m$  often equals 1, 4 or 8 for practical reasons.

## 2.2. Definitions related to CCNx/NDN

The terminology regarding CCNx/NDN used in this document is defined in RFC8793 [17] produced by ICNRG. They are consistent with the relevant documents ([1] [18]).

## 3. CCNx/NDN Basics

We briefly explain the key concepts of CCNx/NDN. In a CCNx/NDN network, there are two types of packets at the network level: interest and data packet (defined in Section 3.4 of [17]). The term of content object, which means a unit of content data, is an alias to data packet [17]. The ICN consumer (defined in Section 3.2 of [17]) requests a content object by sending an interest that carries the name of the data.

Once an ICN forwarder (defined in Section 3.2 of [17]) receives an interest, it performs a series of lookups: first it checks if it has a copy of the requested content object available in the cache storage called Content Store (CS) (defined in Section 3.3 of [17]). If it does, it returns the data, and the transaction is considered to have been successfully completed.

If it does not have a copy of the requested content object in the CS, it performs a lookup of the Pending Interest Table (PIT) (defined in Section 3.3 of [17]) to check if there is already an outgoing interest for the same content object. If there is no such interest, then it creates an entry in the PIT that lists the name included in the interest, and the interfaces from which it received the interest.

This is later used to send the content object back, as interest packets do not carry a source field that identifies the consumer. If there is already a PIT entry for this name, it is updated with the incoming interface of this new interest, and the interest is discarded.

After the PIT lookup, the interest undergoes a Forwarding Information Base (FIB) (defined in Section 3.3 of [17]) lookup for selecting an outgoing interface. The FIB lists name prefixes and their corresponding forwarding interfaces in order to send the interest towards a forwarder that possesses a copy of the requested data.

Once a copy of the data is retrieved, it is sent back to the consumer(s) using the trail of PIT entries; forwarders remove the PIT state every time that an interest is satisfied, and may store the data in their CS.

Data packets carry some information for verifying data integrity and origin authentication, and in particular, that the data is indeed that which corresponds to the name [24]. This is necessary because authentication of the object is crucial in CCNx/NDN. However, this step is optional at forwarders in order to speed up the processing.

The key aspect of CCNx/NDN is that the consumer of the content does not establish a session with a specific server. Indeed, the forwarder or producer (defined in Section 3.2 of [17]) that returns the content object is not aware of the network location of the consumer and the consumer is not aware of the network location of the node that provides the content. This, in theory, allows the interests to follow different paths within a network or even to be sent over completely different networks.

#### 4. NC Basics

While the forwarding node simply relays received data packets in conventional IP communication networks, NC allows the node to combine some data packets that are already received into one or several output packets to be sent. In this section, we simply describe the basic operations of NC. Herein, we focus on RLC in a block coding manner that is well known as a major coding technique.

For simplicity, let us consider an example case of end-to-end coding wherein a producer and consumer respectively perform encoding and decoding for a content object. This end-to-end coding is regarded as a special case of NC. The producer splits the content into several blocks called generations. Encoding and decoding are performed independently on a per-block (per-generation) basis. Let us assume that each generation consists of  $K$  original source packets of the

same size. When the packets do not have the same size, zero padding is added. In order to generate one repair packet within a certain generation, the producer linearly combines  $K$  of the original source packets, where additions and multiplications are performed using a coding vector consisting of  $K$  coding coefficients that are randomly selected in a certain finite field. The producer may respond to interests to send the corresponding source packets and repair packets in the content flow (called systematic coding), where the repair packets are typically used for recovering lost source packets.

Repair packets can also be used for performing encoding. If the forwarding nodes know each coding vector and generation identifier (hereinafter referred to as generation ID) of the received repair packets, they may perform an encoding operation (called re-coding), which is the most distinctive feature of NC compared to other coding techniques.

At the consumer, decoding is performed by solving a set of linear equations that are represented by the coding vectors of the received source and repair packets (possibly only repair packets) within a certain generation. In order to obtain all the source packets, the consumer requires  $K$  linearly independent equations. In other words, the consumer must receive at least  $K$  linearly independent data packets (called innovative packets). As receiving a linearly dependent data packet is not useful for decoding, re-coding should generate and provide innovative packets. One of major benefits of RLC is that even for a small-sized finite field (e.g.,  $q=2^8$ ), the probability of generating linearly dependent packets is negligible [32].

## 5. Advantages of NC and CCNx/NDN

Combining NC and CCNx/NDN can contribute to effective large-scale content/information dissemination. They individually provide similar benefits such as throughput/capacity gain and robustness enhancement. The difference between their approaches is that, the former considers content flow as algebraic information that is to be combined [20], while the latter focuses on the content/information itself at the networking layer. Because these approaches are complementary and their combination would be advantageous, it is natural to combine them.

The name-based communication in CCNx/NDN enables consumers to obtain requested content objects without establishing and maintaining end-to-end communication channels between nodes. This feature facilitates the exploitation of the in-network cache and multipath/multisource retrieval and also supports consumer mobility without the need for updating the location information/identifier during handover

[16]. Furthermore, the name-based communication intrinsically supports multicast communication because identical interests are aggregated at the forwarders.

NC can enable the CCNx/NDN transport system to effectively distribute and cache the data associated with multipath data retrieval [9]. Exploiting multipath data retrieval and in-network caching with NC contributes to not only improving the cache hit rate but also expanding the anonymity set of each consumer (the set of potential routers that can serve a given consumer) [30]. The expansion makes it difficult for adversaries to infer the content consumed by others, and thus contributes to improving cache privacy. Others also have introduced some use cases of the application of NC in CCNx/NDN, such as the cases of content dissemination with in-network caching [10] [13] [14], seamless consumer mobility [11] [36], and low-latency low-loss video streaming [15]. In this context, it is well worth considering NC integration in CCNx/NDN.

## 6. Technical Considerations

This section presents the considerations for CCNx/NDN with NC in terms of network architecture and protocol. This document focuses on NC when employed in a block coding manner.

### 6.1. Content Naming

Naming content objects is as important for CCNx/NDN as naming hosts is in the current-day Internet [24]. In this section, two possible naming schemes are presented.

#### 6.1.1. Unique Naming for NC Packets

Each source and repair packet (hereinafter referred to as NC packet) may have a unique name as each original content object has in CCNx/NDN, as PIT and CS operations typically require a unique name for identifying the NC packet. As a method of naming an NC packet that takes into account the feature of block coding, the coding vector and the generation ID can be used as a part of the content object name. As in [10], when the generation ID is "g-id", generation size is 4, and coding vector is (1,0,0,0), the name could be /CCNx.com/video-A/g-id/1000. Some other identifiers and/or parameters related to the encoding scheme can also be used as name components. For instance, the encoding ID specifying the coding scheme may be used with "enc-id" such as /CCNx.com/video-A/enc-id/g-id/1000, as defined in the FEC Framework (FECFRAME) [26]. This naming scheme is simple and can support the delivery of NC packets with exactly the same operations in the PIT/CS as those for the content objects.

If a content-naming schema such as the one presented above is used, an interest requesting an NC packet may have the full name including a generation ID and coding vector (/CCNx.com/video-A/g-id/1000) or only the name prefix including only a generation ID (/CCNx.com/video-A/g-id). In the former case, exact name matching to the PIT is simply performed at data forwarders (as in CCNx/NDN). The consumer is able to specify and retrieve an innovative packet necessary for the consumer to decode successfully. This could shift the generation of the coding vector from the data forwarder onto the consumer.

In the latter case, partial name matching is required at the data forwarders. As the interest with only the prefix name matches any NC packet with the same prefix, the consumer could immediately obtain an NC packet from a nearby CS (in-network cache) without knowing the coding vectors of the cached NC packets in advance. In the case wherein NC packets in transit are modified by in-network re-coding performed at forwarders, the consumer could also receive the modified NC packets. However, in contrast to the former case, the consumer may fail to obtain sufficient degrees of freedom (see Section 6.2.3). To address this issue, a new TLV type in an interest message may be required for specifying further coding information in order to limit the NC packets to be received. For instance, this is enabled by specifying the coding vectors of innovative packets for the consumer (also called decoding matrix) as in [9]. This extension may incur an interest packet of significantly increased size, and it may thus be useful to use compression techniques for coding vectors [27] [28]. Without such coding information provided by the interest, the forwarder would be required to maintain some records regarding the interest packets that were satisfied previously (See Section 6.2.3).

#### 6.1.2. Non-Unique Naming for NC Packets

An NC packet may have a name that indicates that it is an NC packet, and move the coding information into a metadata field in the payload (i.e., the name includes the data type, source or repair packet). This would not be beneficial for applications or services that may not need to understand the packet payload. Owing to the possibility that multiple NC packets may have the same name, some mechanism is required for the consumer to obtain innovative packets. As described in Section 6.3, a mechanism for managing the multiple innovative packets in the CS would also be required. In addition, extra computational overhead would be incurred when the payload is being encrypted.

## 6.2. Transport

The pull-based request-response feature of CCNx/NDN is a fundamental principle of its transport layer; one interest retrieves at most one data packet. This means that a forwarder or producer cannot inject unrequested data packets on its own initiative. It is believed that it is important that this rule not be violated, as 1) it would open denial-of-service (DoS) attacks, 2) it invalidates existing congestion control approaches following this rule, and 3) it would reduce the efficiency of existing consumer mobility approaches. Thus, the following basic operation should be considered for applying NC to CCNx/NDN. Nevertheless, such security considerations must be addressed if this rule were to be violated.

### 6.2.1. Scope of NC

An open question is whether data forwarder can perform in-network re-coding with data packets that are being received in transit, or if only the data that matches an interest can be subject to NC operations. In the latter case, encoding or re-coding is performed to generate the NC packet at any forwarder that is able to respond to the interest. This could occur when each NC packet has a unique name and interest has the full name. On the other hand, if interest has a partial name without any coding vector information or multiple NC packets have the same name, the former case may occur; re-coding occurs anywhere in the network where it is possible to modify the received NC packet and forward it. As CCNx/NDN comprises mechanisms for ensuring the integrity of the data during transfer, in-network re-coding introduces complexities in the network that needs consideration for the integrity mechanisms to still work. Similarly, in-network caching of NC packets at forwarders may be valuable; however, the forwarders would require some mechanisms to validate the NC packets (see Section 8).

### 6.2.2. Consumer Operation

To obtain NC benefits (possibly associated with in-network caching), the consumer is required to issue interests that direct the forwarder (or producer) to respond with innovative packets if available. In the case where each NC packet may have a unique name (as described in Section 6.1), by issuing an interest specifying a unique name with g-id and the coding vector for an NC packet, the consumer could appropriately receive an innovative packet if it is available at some forwarders.

In order to specify the exact name of the NC packet to be retrieved, the consumer is required to know the valid naming scheme. From a practical viewpoint, it is desirable for the consumer application to

automatically construct the right name components without depending on any application specifications. To this end, the consumer application may retrieve and refer to a manifest [1] that enumerates the content objects including NC packets, or may use some coding scheme specifier as a name component to construct the name components of interests to request innovative packets.

Conversely, the consumer without decoding capability (e.g., specific sensor node) may want to receive only the source packets. As described in Section 6.1, because the NC packet can have a name that is explicitly different from source packets, issuing interests for retrieving source packets is possible.

### 6.2.3. Forwarder Operation

If the forwarder constantly responds to the incoming interests by returning non-innovative packets, the consumer(s) cannot decode and obtain the source packets. This issue could happen when 1) incoming interests for NC packets do not specify some coding parameters such as the coding vectors to be used, and 2) the forwarder does not have a sufficient number of linearly independent NC packets (possibly in the CS) to use for re-coding. In this case, the forwarder is required to determine whether or not it can generate innovative packets to be forwarded to the interface(s) at which the interests arrived. An approach to deal with this issue is that the forwarder maintains a tally of the interests for a specific name, generation ID and the incoming interface(s), in order to record how many degrees of freedom have already been provided [10]. As such a scheme requires state management (and potentially timers) in forwarders, scalability and practicality must be considered. In addition, some transport mechanism for in-network loss detection and recovery [15] [36] at forwarder as well as a consumer-driven mechanism could be indispensable for enabling fast loss recovery and realising NC gains. If a forwarder cannot either return a matching innovative packet from its local content store, nor produce on-the-fly a recoded packet that is innovative, it is important that the forwarder not simply return a non-innovative packet but instead do a forwarding lookup in its FIB and forward the interest toward the producer or upstream forwarder that can provide an innovative packet. In this context, to retrieve innovative packet effectively and quickly, an appropriate setting of the FIB and efficient interest forwarding strategies should also be considered.

In another possible case, when receiving interests only for source packets, the forwarder may attempt to decode and obtain all the source packets and store them (if the full cache capacity are available), thus enabling a faster response to subsequent interests. As re-coding or decoding results in an extra computational overhead,



the forwarder is required to determine how to respond to received interests according to the use case (e.g., a delay-sensitive or delay-tolerant application) and the forwarder situation, such as available cache space and computational capability.

#### 6.2.4. Producer Operation

Before performing NC for specified content in CCNx/NDN, the producer is responsible for splitting the overall content into small content objects to avoid packet fragmentation that could cause unnecessary packet processing and degraded throughput. The size of the content objects should be within the allowable packet size in order to avoid packet fragmentation in CCNx/NDN network. The producer performs the encoding operation for a set of the small content objects, and the naming process for the NC packets.

If the producer takes the lead in determining what coding vectors to use in generating the NC packets, there are three general strategies for naming and producing the NC packets:

1. consumers themselves understand in detail the naming conventions used for NC packets and thereby can send the corresponding interests toward the producer to obtain NC packets whose coding parameters have already been determined by the producer.
2. the producer determines the coding vectors and generates the NC packets after receiving interests specifying the packets the consumer wished to receive.
3. The naming scheme for specifying the coding vectors and corresponding NC packets is explicitly represented via a "Manifest" (e.g., FLIC [23]) that can be obtained by the consumer and used to select among the available coding vectors and their corresponding packets, and thereby send the corresponding interests.

In the first case, although the consumers cannot flexibly specify a coding vector for generating the NC packet to obtain, the latency for obtaining the NC packet is less than in the latter two cases. For the second case, there is a latency penalty for the additional NC operations performed after receiving the interests. For the third case, the NC packets to be included in the manifest must be pre-computed by the producer (since the manifest references NC packets by their hashes, not their names), but the producer can select which to include the manifest, and produce multiple manifests either in advance or on demand with different coding tradeoffs if so desired.

A common benefit the first two approaches to end-to-end coding is that if the producer adds a signature on the NC packets, data validation becomes possible throughout (as is the case with CCNx/NDN operation in the absence of NC). The third approach of using a manifest trades off the additional latency incurred by the need to fetch the manifest against the efficiency of needing a signature only on the manifest and not on each individual NC packet.

#### 6.2.5. Backward Compatibility

NC operations should be applied in addition to the regular ICN behavior, and should function alongside regular ICN operations. Hence, nodes which do not support NC should still be able to properly handle packets, not only in being able to forward the NC packets, but also to cache these packets. An NC framework should be compatible with a regular framework in order to facilitate backward compatibility and smooth migration from one framework to the other.

#### 6.3. In-network Caching

Caching is a useful technique used for improving throughput and latency in various applications. In-network caching in CCNx/NDN essentially provides support at network level and is highly beneficial owing to the involved exploitation of NC for enabling effective multicast transmission [37], multipath data retrieval [10] [11], fast loss recovery [15]. However, there remain several issues to be considered.

There generally exist limitations in the CS capacity, and the caching policy affects the consumer's performance [29] [34] [35]. It is thus crucial for forwarders to determine which content objects should be cached and which discarded. As delay-sensitive applications often do not require an in-network cache for a long period owing to their real-time constraints, forwarders have to know the necessity for caching received content objects to save the caching volume. In CCNx, this could be made possible by setting a Recommended Cache Time (RCT) in the optional header of the data packet at the producer side. The RCT serves as a guideline for the CS cache in determining how long to retain the content object. When the RCT is set as zero, the forwarder recognizes that caching the content object is not useful. Conversely, the forwarder may cache it when the RCT has a greater value. In NDN, the TLV type of FreshnessPeriod could be used.

One key aspect of in-network caching is whether or not forwarders can cache NC packets in their CS. They may be caching the NC packets without having the ability to perform a validation of the content objects. Therefore, the caching of the NC packets would require some mechanism to validate the NC packets (see Section 8). In the case

wherein the NC packets have the same name, it would also require some mechanism to identify them.

#### 6.4. Seamless Consumer Mobility

A key feature of CCNx/NDN is that it is sessionless, which enables the consumer and forwarder to send multiple interests toward different copies of the content in parallel, by using multiple interfaces at the same time in an asynchronous manner. Through the multipath data retrieval, the consumer could obtain the content from multiple copies that are distributed while using the aggregate capacity of multiple interfaces. For the link between the consumer and the multiple copies, the consumer can perform a certain rate adaptation mechanism for video streaming [11] or congestion control for content acquisition [12].

NC adds a reliability layer to CCNx in a distributed and asynchronous manner, because NC provides a mechanism for ensuring that the interests sent to multiple copies of the content in parallel retrieve innovative packets, even in the case of packet losses on some of the paths/networks to these copies. This applies to consumer mobility events [11], wherein the consumer could receive additional degrees of freedom with any innovative packet if at least one available interface exists during the mobility event. An interest forwarding strategy at the consumer (and possibly forwarder) for efficiently obtaining innovative packets would be required for the consumer to achieve seamless consumer mobility.

### 7. Challenges

This section presents several primary challenges and research items to be considered when applying NC in CCNx/NDN.

#### 7.1. Adoption of Convolutional Coding

Several block coding approaches have been proposed thus far; however, there is still not sufficient discussion and application of the convolutional coding approach (e.g., sliding or elastic window coding) in CCNx/NDN. Convolutional coding is often appropriate for situations wherein a fully or partially reliable delivery of continuous data flows is required, and especially when these data flows feature realtime constraints. As in [40], on an end-to-end coding basis, it would be advantageous for continuous content flow to adopt sliding window coding in CCNx/NDN. In this case, the producer is required to appropriately set coding parameters and let the consumer know the information, and the consumer is required to send interests augmented with feedback information regarding the data reception and/or decoding status. As CCNx/NDN utilises hop-by-hop

forwarding state, it would be worth discussing and investigating how convolutional coding can be applied in a hop-by-hop manner and what benefits might accrue. In particular, in the case wherein in-network re-coding could occur at forwarders, both the encoding window and CS management would be required, and the corresponding feasibility and practicality should be considered.

## 7.2. Rate and Congestion Control

The addition of redundancy using repair packets may result in further network congestion and could adversely affect the overall throughput. In particular, in a situation wherein fair bandwidth sharing is more desirable, each streaming flow must adapt to the network conditions to fairly consume the available link bandwidth. It is thus necessary that each content flow cooperatively implement congestion control to adjust the consumed bandwidth [22]. From this perspective, an effective deployment approach (e.g., a forwarder-supported approach that can provide benefits under partial deployment) is required.

As described in Section 6.4, NC can contribute to seamless consumer mobility by obtaining innovative packets without receiving duplicated packets through multipath data retrieval, and avoiding duplicated packets has congestion control benefits as well. It can be challenging to develop an effective rate and congestion control mechanism in order to achieve seamless consumer mobility while improving the overall throughput or latency by fully exploiting NC operations.

## 7.3. Security

While CCNx/NDN introduces new security issues at the networking layer that are different from the IP network, such as a cache poisoning and pollution attacks, a DoS attack using interest packets, some security approaches are already provided [24] [25]. The application of NC in CCNx/NDN brings two potential security aspects that need to be dealt with.

The first is in-network re-coding at forwarders. Some mechanism for ensuring the integrity of the NC packets newly produced by in-network re-coding is required in order for consumers or other forwarders to receive valid NC packets. To this end, there are some possible approaches described in Section 8, but there may be more effective method with lower complexity and computation overhead.

The second is that attackers maliciously request and inject NC packets, which could amplify some attacks. As NC packets are unpopular in general use, they could be targeted by a cache pollution attack that requests less popular content objects more frequently to

undermine popularity-based caching by skewing the content popularity. Such an attack needs to be dealt with in order to maintain the in-network cache efficiency. By injecting invalid NC packets with the goal of filling the CSs at the forwarders with them, the cache poisoning attack could be effectual depending on the exact integrity coverage on NC packets. On the assumption that each NC packet has the valid signature, the straightforward approach would comprise the forwarders verifying the signature within the NC packets in transit and only transmitting and storing the validated NC packets. However, as performing a signature verification by the forwarders may be infeasible at line speed, some mechanisms should be considered for distributing and reducing the load of signature verification, in order to maintain in-network cache benefits such as latency and network-load reduction.

#### 7.4. Routing Scalability

In CCNx/NDN, a name-based routing protocol without a resolution process streamlines the routing process and reduces the overall latency. In IP routing, the growth in the routing table size has become a concern. It is thus necessary to use a hierarchical naming scheme in order to improve the routing scalability by enabling the aggregation of the routing information.

To realize the benefits of NC, consumers need to efficiently obtain innovative packets using multipath retrieval mechanisms of CCNx/NDN. This would require some efficient routing mechanism to appropriately set the FIB and also an efficient interest forwarding strategy. Such routing coordination may create routing scalability issues. It would be challenging to achieve effective and scalable routing for interests requesting NC packets as well as to simplify the routing process.

#### 8. Security Considerations

In-network re-coding is a distinguishing feature of NC. Only valid NC packets produced by in-network re-coding must be requested and utilized (and possibly stored). To this end, there exist some possible approaches. First, as a signature verification approach, the exploitation of multi-signature capability could be applied. This allows not only the original content producer but also some forwarders responsible for in-network re-coding to have their own unique signing key. Each forwarder of the group signs newly generated NC packet in order for other nodes to be able to validate the data with the signature. The CS may verify the signature within the NC packet before storing it to avoid invalid data caching. Second, as a consumer-dependent approach, the consumer puts a restriction on the matching rule using only the name of the requested

data. The interest ambiguity can be clarified by specifying both the name and the key identifier (the producer's public key digest) used for matching to the requested data. This KeyId restriction is built in the CCNx design [1]. Only the requested data packet satisfying the interest with the KeyId restriction would be forwarded and stored in the CS, thus resulting in a reduction in the chances of cache poisoning. Moreover, in the CCNx design, there exists the rule that the CS obeys in order to avoid amplifying invalid data; if an interest has a KeyID restriction, the CS must not reply unless it knows that the signature on the matching content object is correct. If the CS cannot verify the signature, the interest may be treated as a cache miss and forwarded to the upstream forwarder(s). Third, as a certificate chain management approach (possibly without certificate authority), some mechanism such as [31] could be used to establish a trustworthy data delivery path. This approach adopts the hop-by-hop authentication mechanism, wherein forwarding-integrated hop-by-hop certificate collection is performed to provide suspension certificate chains such that the data retrieval is trustworthy.

Depending on the adopted caching strategy such as cache replacement policies, forwarders should also take caution when storing and retaining the NC packets in the CS as they could be targeted by cache pollution attacks. In order to mitigate the cache pollution attacks' impact, forwarders should check the content request frequencies to detect the attack and may limit requests by ignoring some of the consecutive requests. The forwarders can then decide to apply or change to the other cache replacement mechanism.

The forwarders or producers require careful attention to the DoS attacks aiming at provoking the high load of NC operations by using the interests for NC packets. In order to mitigate such attacks, the forwarders could adopt a rate-limiting approach. For instance, they could monitor the PIT size growth for NC packets per content to detect the attacks, and limit the interest arrival rate when necessary. If the NC application wishes to secure an interest (considered as the NC actuator) in order to prevent such attacks, the application should consider using an encrypted wrapper and an explicit protocol.

## 9. Acknowledgements

The authors would like to thank ICNRG and NWCRG members, especially Marie-Jose Montpetit, David Oran, Vincent Roca, and Thierry Turlletti, for their valuable comments and suggestions on this document.

## 10. Informative References

- [1] Mosko, M. and et al., "Content-Centric Networking (CCNx) Semantics", RFC 8569, July 2019, <<https://tools.ietf.org/html/rfc8569>>.
- [2] Gkantsidis, C. and P. Rodriguez, "Cooperative Security for Network Coding File Distribution", Proc. Infocom, IEEE, April 2006.
- [3] Cai, N. and R. Yeung, "Secure network coding", Proc. International Symposium on Information Theory (ISIT), IEEE, June 2002.
- [4] Lima, L., Gheorghiu, S., Barros, J., Medard, M., and A. Toledo, "Secure Network Coding for Multi-Resolution Wireless Video Streaming", IEEE Journal of Selected Area (JSAC), vol. 28, no. 3, April 2010.
- [5] Vilea, J., Lima, L., and J. Barros, "Lightweight security for network coding", Proc. ICC, IEEE, May 2008.
- [6] Dimarkis, A., Godfrey, P., Wu, Y., Wainwright, M., and K. Ramchandran, "Network Coding for Distributed Storage Systems", IEEE Trans. Information Theory, vol. 56, no.9, September 2010.
- [7] Gkantsidis, C. and P. Rodriguez, "Network coding for large scale content distribution", Proc. Infocom, IEEE, March 2005.
- [8] Seferoglu, H. and A. Markopoulou, "Opportunistic Network Coding for Video Streaming over Wireless", Proc. Packet Video Workshop (PV), IEEE, November 2007.
- [9] Montpetit, M., Westphal, C., and D. Trossen, "Network Coding Meets Information-Centric Networking: An Architectural Case for Information Dispersion Through Native Network Coding", Proc. Workshop on Emerging Name-Oriented Mobile Networking Design (NoM), ACM, June 2012.
- [10] Saltarin, J., Bourtsoulatze, E., Thomos, N., and T. Braun, "NetCodCCN: a network coding approach for content-centric networks", Proc. Infocom, IEEE, April 2016.

- [11] Ramakrishnan, A., Westphal, C., and J. Saltarin, "Adaptive Video Streaming over CCN with Network Coding for Seamless Mobility", Proc. International Symposium on Multimedia (ISM), IEEE, December 2016.
- [12] Mahdian, M., Arianfar, S., Gibson, J., and D. Oran, "MIRCC: Multipath-aware ICN Rate-based Congestion Control", Proc. Conference on Information-Centric Networking (ICN), ACM, September 2016.
- [13] Wang, J., Ren, J., Lu, K., Wang, J., Liu, S., and C. Westphal, "An Optimal Cache Management Framework for Information-Centric Networks with Network Coding", Proc. Networking Conference, IFIP/IEEE, June 2014.
- [14] Wang, J., Ren, J., Lu, K., Wang, J., Liu, S., and C. Westphal, "A Minimum Cost Cache Management Framework for Information-Centric Networks with Network Coding", Computer Networks, Elsevier, August 2016.
- [15] Matsuzono, K., Asaeda, H., and T. Turletti, "Low Latency Low Loss Streaming using In-Network Coding and Caching", Proc. Infocom, IEEE, May 2017.
- [16] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", Proc. CoNEXT, ACM, December 2009.
- [17] Wissingh, B. and et al., "Information-Centric Networking (ICN): Content-Centric Networking (CCNx) and Named Data Networking (NDN) Terminology", RFC 8793, June 2020, <<https://tools.ietf.org/html/rfc8793>>.
- [18] Mosko, M. and et al., "Content-Centric Networking (CCNx) Messages in TLV Format", RFC 8609, July 2019, <<https://tools.ietf.org/html/rfc8609>>.
- [19] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Claffy, K., Crowley, P., Papadopoulos, C., Wang, L., and B. Zhang, "Named data networking", ACM Comput. Commun. Rev., vol. 44, no. 3, July 2014.
- [20] Koetter, R. and M. Medard, "An Algebraic Approach to Network Coding", IEEE/ACM Trans. on Networking, vol. 11, no 5, Oct. 2003.



- [21] Adamson, B. and et al., "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, June 2018, <<https://tools.ietf.org/html/rfc8406>>.
- [22] Kuhn, N., Lochin, E., Michel, F., and M. Welzl, "Coding and Congestion Control in Transport", Work in Progress, draft-irtf-nwcrg-coding-and-congestion-09, June 2021.
- [23] Tschudin, C., Wood, C., Mosko, M., and D. Oran, "File-Like ICN Collections (FLIC)", Work in Progress, draft-irtf-icnrg-flic-03, Nov. 2021.
- [24] Kutscher, D. and et al., "Information-Centric Networking (ICN) Research Challenges", RFC 7927, July 2016.
- [25] Pentikousis, K. and et al., "Information-Centric Networking: Evaluation and Security Considerations", RFC 7945, July 2019.
- [26] Watson, M. and et al., "Forward Error Correction (FEC) Framework", RFC 6363, Oct. 2011.
- [27] Thomos, N. and P. Frossard, "Toward one Symbol Network Coding Vectors", IEEE Communications letters, vol. 16, no. 11, November 2012.
- [28] Lucani, D., Pedersen, M., Heide, J., and F. Fitzek, "Fulcrum Network Codes: A Code for Fluid Allocation of Complexity", available at <http://arxiv.org/abs/1404.6620>, April 2014.
- [29] Perino, D. and M. Varvello, "A reality check for content centric networking", Proc. SIGCOMM Workshop on Information-centric networking (ICN'11), ACM, August 2011.
- [30] Wu, Q., Li, Z., Tyson, G., Uhlig, S., Kaafar, M., and G. Xie, "Privacy-Aware Multipath Video Caching for Content-Centric Networks", IEEE Journal of Selected Area (JSAC) vol. 38, no. 8, June 2016.
- [31] Li, R., Asaeda, H., and J. Wu, "DCAuth: Data-Centric Authentication for Secure In-Network Big-Data Retrieval", IEEE Trans. on Network Science and Engineering vol. 7, no. 1, September 2018.
- [32] Wu, Y., Chou, P., and K. Jain, "A comparison of network coding and tree packing", Proc. ISIT, IEEE, June 2004.

- [33] Ho, T., Medard, M., Koetter, R., Karger, R., Effros, D., Shi, M., and B. Leong, "A Random Linear Network Coding Approach to Multicast", IEEE Trans. Information Theory, vol. 52, no.10, October 2006.
- [34] Podlipnig, S. and L. Osz, "A Survey of Web Cache Replacement Strategies", Proc. ACM Computing Surveys vol. 35, no. 4, December 2003.
- [35] Rossini, G. and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies", Elsevier Computer Communication, vol.36, no. 7, April 2013.
- [36] Carofiglio, G., Muscariello, L., Papalini, M., Rozhnova, N., and X. Zeng, "Leveraging ICN In-network Control for Loss Detection and Recovery in Wireless Mobile networks", Proc. ICN ACM, September 2016.
- [37] Ali, M. and U. Niesen, "Coding for Caching: Fundamental Limits and Practical Challenges", IEEE Communications Magazine vol. 54, no. 8, August 2016.
- [38] Koetter, R. and F. Kschischang, "An algebraic approach to network coding", IEEE Trans. Netw. vol.11, no.5, October 2003.
- [39] Vyetenko, S., Ho, T., Effros, M., Klierer, J., and E. Erez, "Rate regions for coherent and noncoherent multisource network error correction", Proc. International Symposium on Information Theory (ISIT), IEEE, July 2009.
- [40] Tournoux, P., Lochin, E., Lacan, J., Bouabdallah, A., and V. Roca, "On-the-Fly Erasure Coding for Real-Time Video Applications", IEEE Trans. Multimedia vol.13, no.4, August 2011.

#### Authors' Addresses

Kazuhisa Matsuzono  
National Institute of Information and Communications Technology  
4-2-1 Nukui-Kitamachi  
Koganei, Tokyo 184-8795  
Japan

Email: matsuzono@nict.go.jp

Hitoshi Asaeda  
National Institute of Information and Communications Technology  
4-2-1 Nukui-Kitamachi  
Koganei, Tokyo 184-8795  
Japan

Email: asaeda@nict.go.jp

Cedric Westphal  
Huawei  
2330 Central Expressway  
Santa Clara, California 95050  
USA

Email: cedric.westphal@futurewei.com,

NWCRG  
Internet-Draft  
Intended status: Informational  
Expires: May 20, 2020

V. Roca (Ed.)  
INRIA  
J. Detchart  
ISAE - Supaero  
C. Adjih  
INRIA  
M. Pedersen  
Steinwurf ApS  
November 17, 2019

Generic Application Programming Interface (API) for Sliding Window FEC  
Codes  
draft-roca-nwcrg-generic-fec-api-07

Abstract

This document introduces a generic Application Programming Interface (API) for sliding window FEC codes. This API is meant to be compatible with any sliding window FEC code. It defines the core procedures and functions meant to control the codec (i.e., implementation of the FEC code). However, it leaves out all upper layer aspects that are the responsibility of the application or protocol making use of the codec. As a consequence, this is not an API for a FEC Scheme since certain mechanisms that must be defined by any FEC Scheme (e.g., signalling and FEC Payload IDs) are the responsibility of the caller instead of being addressed by the codec. A first goal of this document is to pave the way for a future open-source implementation of such codes, another goal is to simplify the development of content delivery protocols that rely on sliding window FEC codes for robust transmissions.

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 May 20, 2020.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions and Abbreviations . . . . .	3
3. AL-FEC Codes and Mechanisms Considered by the Generic API . .	3
3.1. Mechanisms Considered or Ignored by the API . . . . .	5
4. Generic API for Sliding Window FEC Codes . . . . .	6
4.1. General Definitions Common to the Encoder and Decoder . .	6
4.2. Encoder . . . . .	9
4.3. Decoder . . . . .	13
4.4. Coding Window Functions at an Encoder and Decoder . . . .	17
4.5. Coding Coefficients Functions at an Encoder and Decoder .	19
5. Security Considerations . . . . .	22
6. IANA Considerations . . . . .	22
7. Acknowledgments . . . . .	23
8. References . . . . .	23
8.1. Normative References . . . . .	23
8.2. Informative References . . . . .	23
Authors' Addresses . . . . .	23

## 1. Introduction

Forward Erasure Correction (FEC) codes are a key element of communication systems, used to efficiently recover from packet losses during content delivery sessions. Among the FEC codes working at the network and higher layers, one can broadly distinguish block codes and sliding window codes. Block FEC codes require the data flow coming from the application to be segmented into blocks of a predefined maximum size, before generating a certain number of repair packets. With the second type of FEC codes, an encoding window continuously slides over the set of source data and repair packets are generated at any time by computing for instance a linear combination of data present in the encoding window. This fundamental

difference seriously impacts the way they can be used by a content delivery protocol or application.

This document introduces a generic Application Programming Interface (API) for sliding window FEC codes. This API is meant to be usable by any sliding window FEC code and FEC Scheme independently of the protocol that may rely on it. This API defines the core procedures and functions meant to control the codec (i.e., implementation of the FEC code), but leaves out all upper layer aspects that are the responsibility of the application making use of the codec.

This API is meant to be usable by any sliding window FEC code. independently of the FEC Scheme or network coding protocol that may rely on it This API defines the core procedures and functions meant to control the codec (i.e., implementation of the FEC code), but leaves out all upper layer aspects that are the responsibility of the application making use of the codec. For instance, those restricted to end-to-end use-cases as well as those compatible with in-network re-encoding use-cases. Additionally, this API is not impacted by the intra-flow versus inter-flow nature of the use-case, nor is it impacted by the single-path versus multi-paths nature of the use-case, since those are usage considerations under the responsibility of the caller.

A goal of this document is to pave the way for a future open-source implementation of such codes.

## 2. Definitions and Abbreviations

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].

This document uses the following definitions and abbreviations:

XXX

## 3. AL-FEC Codes and Mechanisms Considered by the Generic API

This generic FEC API is meant to be used with:

- o sliding window codes, that manage an encoding window (of fixed or variable size) that slides over the set of source symbols at the sender. On the opposite, block codes (e.g., Reed-Solomon, LDPC, Raptor(Q)) are out of scope;
- o codes that are restricted to use-cases that involve a single encoding point and a single decoding point (i.e., FEC operations are carried out either within the end-hosts or middle-boxes), as

- well as codes that can be used with use-cases that involve in-network re-coding operations;
- o use-cases that are limited to an intra-flow coding (simple case), as well as use-cases that involve inter-flow coding. This second case is more complex to address (e.g., with questions such as how to identify a packet of a flow?) however this is the responsibility of the application or protocol using this codec and not the codec itself. This aspect is therefore transparent to the API;
  - o use-cases that are limited to single-path communications and use-cases that consider multi-path communications. Here also this is a usage consideration that is transparent to the API;
  - o use-cases that involve a dynamic adaptation of the codec parameters (e.g., its code rate because the communication path losses is known thanks to feedbacks and an appropriate strategy can be defined);
  - o fixed code rate or not FEC codes, including rateless codes where the number of repair symbols that can be generated is huge (in theory unlimited);
  - o ideal (MDS) or non-ideal (non-MDS) codes. However most of the time, sliding window codes are non-ideal codes, meaning that slightly more than 1 repair symbols may be required to recover all the 1 lost source symbols;

A key question is to determine what mechanisms are included in the codec and what mechanisms are left to the responsibility of the caller (i.e., an application or a protocol making use of this codec) (Figure 1). More precisely, an FEC Scheme (such as the RLC FEC Scheme [RLC] in case of FECFRAME [fecframe-ext]) defines all the internal code details in order to enable interoperable implementations, but also signaling considerations that are essential to use them in a specific context.

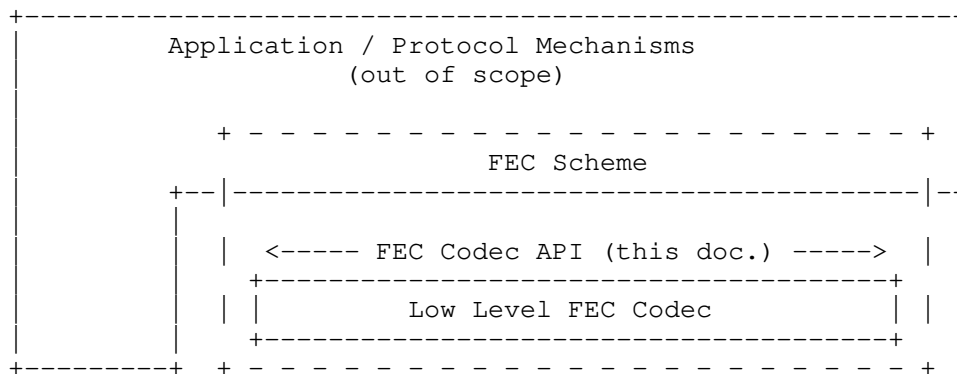


Figure 1: Position of the FEC Codec API with respect to the low level FEC Codec, the FEC Scheme, the protocol and other caller services.

### 3.1. Mechanisms Considered or Ignored by the API

Applying FEC coding, through an FEC Scheme, in a given protocol to improve transmission robustness involves many mechanisms. However, these mechanisms are not all the responsibility of the codec and can be implemented within the application or within the protocol that uses this FEC codec. For instance, the following mechanisms are considered **\*\*out of scope of the API\*\***, being implemented by the caller, without any impact on the codec:

- o memory management;
- o packet transmission and reception;
- o signaling header creation / parsing;
- o ADU to source symbol mapping;
- o code rate adjustment, for instance thanks to the knowledge of losses at a receiver via feedbacks;
- o selective ACK creation and parsing;
- o congestion control.

The following mechanisms are **\*\*within scope of the API\*\***:

- o session management (sender and receiver);
- o encoding window management (sender and receiver);
- o set/get/generate coding coefficients (sender and receiver);
- o build coded symbol (sender only);
- o decode with newly received source or repair symbol (receiver only);



#### 4. Generic API for Sliding Window FEC Codes

The following sections describe the generic API, following a C-language formalism. This API tries to adhere to C99 version of C, although it may not strictly be guaranteed. Everything is prefixed by "swif" (sliding window FEC).

##### 4.1. General Definitions Common to the Encoder and Decoder

This section gathers general definitions that are used by both an encoder and decoder.

About FEC Codepoints:

An application first needs to negotiate with its remote side the right FEC Scheme to use. This negotiation usually relies on the FEC Encoding ID associated to this FEC Scheme for this application. A difficulty is that the FEC Encoding ID space, associated to an IANA registry, is protocol specific and the same value are usually associated to different FEC Schemes depending on the protocol. For instance, the FEC Encoding ID value 2 may be used for two totally different FEC Schemes in protocol A and protocol B. Therefore, the FEC Encoding ID, from the Generic FEC API point of view, cannot be used to uniquely identify the target codec.

The use of a codepoint to identify locally the right FEC codec requires that the application knows a mapping between the FEC Encoding ID it uses for a given protocol, and the local FEC Codepoints corresponding to available codecs. This will be done at development time, the FEC API header file giving access to the `swif_codepoint_t` enumeration with the list of all codecs available locally.

<CODE BEGINS>

```
/**
 * Return value of any function.
 *
 * SWIF_STATUS_OK = 0      Success
 * SWIF_STATUS_FAILURE    Failure. The function called did not succeed to
 *                          perform its task, however this is not an error
 *                          (e.g., it happens when decoding fails).
 * SWIF_STATUS_ERROR      Generic error type. The detailed error type is
 *                          stored in the errno variable of swif_encoder_t and
 *                          swif_decoder_t structures.
 */
typedef enum {
    SWIF_STATUS_OK = 0,
    SWIF_STATUS_FAILURE,
```

```
        SWIF_STATUS_ERROR
    } swif_status_t;

/**
 * Potential errors.
 */
typedef enum {
    SWIF_ERRNO_NULL = 0,                /* everything is fine */
    SWIF_ERRNO_UNSUPPORTED_CODEPOINT,
    /* and many more... */
} swif_errno_t;

/**
 * FEC Codepoints.
 * These identifiers are opaque identifiers that fully identify an FEC
 * code locally, including certain parameters like its Galois Field.
 * These codepoints are codec specific and only have a local meaning.
 * They should not be transmitted as different implementations may use
 * them inconsistently.
 * Note that the same FEC code may be used by several FEC Encoding IDs
 * and therefore share the same codepoint. On the opposite multiple
 * implementations of a given FEC code may exist locally, for instance
 * with different optimizations, and then several codepoints, one per
 * codec, will exist for the same FEC code. The following names are
 * therefore only provided as examples.
 */
typedef enum {
    SWIF_CODEPOINT_NULL = 0,            /* codepoint 0 is reserved */

    /* codepoint for sliding window codec AAA. */
    SWIF_CODEPOINT_AAA_CODEC,

    /* codepoint for sliding window codec BBB. */
    SWIF_CODEPOINT_BBB_CODEC,

    /* list here other identifiers for any codec of interest... */
} swif_codepoint_t;

/**
 * Encoding Symbol Identifier (ESI) generic type.
 * With Sliding Window FEC codes, an ESI is in fact a source symbol
 * identifier, unlike block FEC codes.
 */
typedef uint32_t      esi_t;
```

```

/**
 * Throughout the API, a pointer to this structure is used as an
 * identifier of the encoder instance (or "enc").
 *
 * This generic structure is meant to be extended by each codec with
 * new pieces of information that are specific to each codec.
 */
typedef struct swif_encoder {
    swif_codepoint_t      codepoint;

    /* when a function returns with SWIF_STATUS_ERROR, the errno
     * variable contains a more detailed error type. This variable
     * is set by the codec and accessible to the application in
     * READ ONLY mode. Otherwise its value is undefined. */
    swif_errno_t          swif_errno;

    /* pointers to codec specific versions of API functions. */
    swif_status_t  (*set_callback_functions) (
        struct swif_encoder*, void (*) (void*, esi_t), void*);
    swif_status_t  (*set_parameters) (
        struct swif_encoder*, uint32_t, uint32_t, void*);
    swif_status_t  (*get_parameters) (
        struct swif_encoder*, uint32_t, uint32_t, void*);
    swif_status_t  (*build_repair_symbol) (
        struct swif_encoder*, void*);
    swif_status_t  (*reset_coding_window) (struct swif_encoder*);
    swif_status_t  (*add_source_symbol_to_coding_window) (
        struct swif_encoder*, void*, esi_t);
    swif_status_t  (*remove_source_symbol_from_coding_window) (
        struct swif_encoder*, esi_t);
    swif_status_t  (*get_coding_window_information) (
        struct swif_encoder*, esi_t*, esi_t*, uint32_t*);
    swif_status_t  (*set_coding_coefs_tab) (
        struct swif_encoder*, void*, uint32_t);
    swif_status_t  (*generate_coding_coefs) (
        struct swif_encoder*, uint32_t, uint32_t);
    swif_status_t  (*get_coding_coefs_tab) (
        struct swif_encoder*, void**, uint32_t*);
} swif_encoder_t;

/**
 * Decoder structure that contains whatever is needed for decoding.
 * The exact content of this structure is FEC code dependent, the
 * structure below being a non normative example.
 */
typedef struct swif_decoder {
    swif_codepoint_t      codepoint;

```

```

/* when a function returns with SWIF_STATUS_ERROR, the errno
 * variable contains a more detailed error type. This variable
 * is set by the codec and accessible to the application in
 * READ ONLY mode. Otherwise its value is undefined. */
swif_errno_t          swif_errno;

/* pointers to codec specific versions of API functions. */
swif_status_t  (*set_callback_functions) (
    struct swif_decoder*, void (*) (void*, esi_t),
    void* (*) (void*, esi_t),
    void* (*) (void*, void*, esi_t), void*);
swif_status_t  (*set_parameters) (
    struct swif_decoder*, uint32_t, uint32_t, void*);
swif_status_t  (*get_parameters) (
    struct swif_decoder*, uint32_t, uint32_t, void*);
swif_status_t  (*decode_with_new_source_symbol) (
    struct swif_decoder*, void* const, esi_t);
swif_status_t  (*decode_with_new_repair_symbol) (
    struct swif_decoder*, void* const);
swif_status_t  (*reset_coding_window) (swif_encoder_t*);
swif_status_t  (*add_source_symbol_to_coding_window) (
    struct swif_decoder*, esi_t);
swif_status_t  (*remove_source_symbol_from_coding_window) (
    struct swif_decoder*, esi_t);
swif_status_t  (*set_coding_coefs_tab) (
    struct swif_decoder*, void*, uint32_t);
swif_status_t  (*generate_coding_coefs) (
    struct swif_decoder*, uint32_t, uint32_t);
} swif_decoder_t;
<CODE ENDS>

```

General definitions.

#### 4.2. Encoder

```

<CODE BEGINS>
/**
 * Create and initialize an encoder, providing only key parameters.
 *
 * @param codepoint      opaque identifier that fully identifies the FEC
 *                        code to use.
 * @param verbosity      print information on the codec processing.
 *                        0 is the minimum verbosity, the maximum verbosity
 *                        level being implementation specific.
 * @param symbol_size    source and repair symbol size in bytes. Cannot
 *                        change during the codec instance lifetime.
 * @param max_encoding_window_size
 * @return               pointer to a swif_encoder_t structure if okay, or

```

```
*          NULL in case of error.
**/
swif_encoder_t* swif_encoder_create (
                                swif_codepoint_t codepoint,
                                uint32_t          verbosity,
                                uint32_t          symbol_size,
                                uint32_t          max_coding_window_size);

/**
 * Release an encoder and its associated ressources.
 **/
swif_status_t  swif_encoder_release (swif_encoder_t*      enc);

/**
 * Set the various callback functions for this encoder.
 * All the callback functions require an opaque context parameter, that
 * must be initialized accordingly by the application, since it is
 * application specific.
 *
 * @param enc
 * @param source_symbol_removed_from_coding_window_callback
 *      (IN) Pointer to the function, within the application,
 *      that needs to be called each time a source symbol is
 *      removed from the left side of the coding window.
 *      This callback is called each time the encoding window
 *      slides to the right and an old source symbol needs to
 *      be removed on the left. The application therefore knows
 *      this source symbol will no longer be used by the codec
 *      and can free the associated buffer if need be. This
 *      function does not return anything.
 * @param context_4_callback
 *      (IN) Pointer to the application-specific context that
 *      will be passed to the callback function (if any). This
 *      context is not interpreted by this function.
 * @return
 */
swif_status_t  swif_encoder_set_callback_functions (
                                swif_encoder_t*      enc,
                                void (*source_symbol_removed_from_coding_window_callback) (
                                                                    void*      context,
                                                                    esi_t      old_symbol_esi),
                                void* context_4_callback);

/**
 * This function sets one or more FEC codec specific parameters,
 * using a type/length/value approach for maximum flexibility.

```

```
*
* @param enc
* @param type      (IN) Type of parameter.
* @param length    (IN) length of the pointed value.
* @param value     (IN) Pointer to the value. The exact type of
*                  the object pointed is FEC codec specific.
* @return
*/
swif_status_t swif_encoder_set_parameters (
    swif_encoder_t* enc,
    uint32_t      type,
    uint32_t      length,
    void*         value);

/**
* This function gets one or more FEC codec specific parameters,
* using a type/length/value approach for maximum flexibility.
*
* @param enc
* @param type      (IN) Type of parameter.
* @param length    (IN) length of the pointed value.
* @param value     (IN/OUT) Pointer to the value. The exact type of
*                  the object pointed is FEC codec specific.
*                  This function updates the value object
*                  accordingly. The caller, who knows the FEC codec,
*                  is responsible to allocate the appropriate
*                  object buffer.
* @return
*/
swif_status_t swif_encoder_get_parameters (
    swif_encoder_t* enc,
    uint32_t      type,
    uint32_t      length,
    void*         value);

/**
* List here the FEC codec specific control parameters.
*/
enum {
    swif_ENCODER_GET_PARAM_ENCODER_STATISTICS = 1,
    swif_ENCODER_SET_PARAM_RLC_DENSITY_THRESHOLD
};

/**
* Create a single repair symbol (i.e. perform an encoding).
* Upon return of this function, the application has full control of the
* buffer and is in charge of freeing it when appropriate.
```

```

*
* @param new_buf      (IN) The pointer to the buffer for the repair
*                      symbol to build can either point to a buffer
*                      allocated by the application and initialized to
*                      zero, or let to NULL meaning that this function
*                      will allocate memory.
* @return
*/
swif_status_t  swif_build_repair_symbol (
                                swif_encoder_t* enc,
                                void*          new_buf);
/* FIX ME: must be void** to enable returning a pointer to buffer! */
<CODE ENDS>

```

## Encoder API proposal

```

<CODE BEGINS>
/**
 * Encoder structure that contains whatever is needed for encoding.
 * The exact content of this structure is FEC code dependent, the
 * structure below being a non normative example.
 * However it MUST be aligned with swif_encoder_t (same first items) in
 * order to be able to cast a pointer to one of the two structures,
 * depending on the context.
 */
typedef struct swif_encoder_internal {
    /* generic part of any control block. MUST be first in structure */
    swif_encoder_t  gen;

    /* desired verbosity: 0 is the minimum verbosity, the maximum
     * level being implementation specific. */
    uint32_t        verbosity;

    /* maximum number of source symbols used for any repair symbol */
    uint32_t        max_coding_window_size;

    /* exact size (in bytes) of any source or repair symbol */
    uint32_t        symbol_size;

    /* add whatever may be needed hereafter... */
} swif_encoder_internal_t;

```

Non normative example of internal structure used by an encoder.

## 4.3. Decoder

&lt;CODE BEGINS&gt;

```

/**
 * Create and initialize a decoder, providing only key parameters.
 *
 * @param codepoint      opaque identifier that fully identifies the FEC
 *                        code to use.
 * @param verbosity      print information on the codec processing.
 *                        0 is the minimum verbosity, the maximum verbosity
 *                        level being implementation specific.
 * @param symbol_size    source and repair symbol size in bytes. Cannot
 *                        change during the codec instance lifetime.
 * @param max_coding_window_size
 * @param max_linear_system_size
 * @return               pointer to a swif_decoder_t structure if okay, or
 *                        NULL in case of error.
 */
swif_decoder_t* swif_decoder_create (
                                swif_codepoint_t codepoint,
                                uint32_t         verbosity,
                                uint32_t         symbol_size,
                                uint32_t         max_coding_window_size,
                                uint32_t         max_linear_system_size);

/**
 * Release a decoder and its associated ressources.
 *
 * @param dec            context (i.e., pointer to decoder structure).
 */
swif_status_t  swif_decoder_release (swif_decoder_t*      dec);

/**
 * Set the various callback functions for this decoder.
 * All the callback functions require an opaque context parameter, that
 * must be initialized accordingly by the application, since it is
 * application specific.
 *
 * @param dec            context (i.e., pointer to decoder structure).
 * @param source_symbol_removed_from_linear_system_callback
 *                        (IN) Pointer to the function, within the application, that
 *                        needs to be called each time a source symbol is removed from
 *                        the left side of the linear system.
 *                        This callback is called each time the linear system slides
 *                        to the right and an old source symbol needs to be removed
 *                        on the left. This function does not return anything.

```



```

* @param decodable_source_symbol_callback
*      (IN) Pointer to the function, within the application, that
*      needs to be called each time a source symbol is decodable.
*      What it does is application-dependent, but it MUST return
*      either a pointer to a data buffer, left uninitialized, of
*      the appropriate size, or NULL if the application prefers to
*      let the codec allocate the buffer.
*      In any case the codec is responsible for storing the actual
*      symbol value within the data buffer. Also, no matter
*      whether the data buffer is allocated by the application or
*      the codec, it is the responsibility of the application to
*      free this buffer when needed, once decoding is over (but
*      not before since the codec does not keep any internal copy).
* @param decoded_source_symbol_callback
*      (IN) Pointer to the function, within the application, that
*      needs to be called each time a source symbol is decodable and
*      all computations performed (i.e., the buffer does contain the
*      symbol value).
*      This callback is called in a second time, when the newly
*      decodable source symbol is actually decoded and ready,
*      i.e., when all the computations (like XOR and GF(2**8)
*      operations) have been performed. In any case, it is the
*      responsibility of the application to free this buffer when
*      needed, once decoding is over (but not before since the
*      codec does not keep any internal copy). This function does
*      not return anything.
* @param context_4_callback
*      (IN) Pointer to the application-specific context that will be
*      passed to the callback function (if any). This context is not
*      interpreted by this function.
* @return
*/
swif_status_t swif_decoder_set_callback_functions (
    swif_decoder_t* dec,
    void (*source_symbol_removed_from_linear_system_callback) (
        void* context,
        esi_t old_symbol_esi),
    void* (*decodable_source_symbol_callback) (
        void* context,
        esi_t esi),
    void (*decoded_source_symbol_callback) (
        void* context,
        void* new_symbol_buf,
        esi_t esi),
    void* context_4_callback);

/**

```

```

* This function sets one or more FEC codec specific parameters,
*     using a type/length/value approach for maximum flexibility.
*
* @param dec      context (i.e., pointer to decoder structure).
* @param type     (IN) Type of parameter.
* @param length   (IN) length of the pointed value.
* @param value    (IN) Pointer to the value. The exact type of
*                 the object pointed is FEC codec specific.
* @return
*/
swif_status_t    swif_decoder_set_parameters (
                                swif_decoder_t* dec,
                                uint32_t      type,
                                uint32_t      length,
                                void*         value);

/**
* This function gets one or more FEC codec specific parameters,
* using a type/length/value approach for maximum flexibility.
*
* @param dec      context (i.e., pointer to decoder structure).
* @param type     (IN) Type of parameter.
* @param length   (IN) length of the pointed value.
* @param value    (IN/OUT) Pointer to the value. The exact type of
*                 the object pointed is FEC codec specific.
*                 This function updates the value object
*                 accordingly. The caller, who knows the FEC codec,
*                 is responsible to allocate the appropriate
*                 object buffer.
* @return
*/
swif_status_t    swif_decoder_get_parameters (
                                swif_decoder_t* dec,
                                uint32_t      type,
                                uint32_t      length,
                                void*         value);

/**
* List here the FEC codec specific control parameters.
*/
enum {
    swif_DECODER_GET_PARAM_DECODER_STATISTICS = 1,
    swif_DECODER_SET_PARAM_RLC_DENSITY_THRESHOLD
};

/**
* Submit a received source symbol and try to progress in the decoding.

```

```

* For each decoded source symbol (if any), the application is informed
* through the dedicated callback functions.
*
* This function usually returns SWIF_STATUS_OK, regardless of whether
* this new symbol enabled the decoding of one or several source symbols,
* or SWIF_STATUS_ERROR. It cannot return SWIF_STATUS_FAILURE.
*
* @param dec    context (i.e., pointer to decoder structure).
* @param new_symbol_buf
*             (IN) Pointer to the new source symbol now available (i.e.
*             a new symbol received by the application, or a decoded
*             symbol in case of a recursive call if it makes sense).
* @param new_symbol_esi
*             (IN) encoding symbol ID of the new source symbol.
* @return
*/
swif_status_t    swif_decoder_decode_with_new_source_symbol (
                                swif_decoder_t* dec,
                                void* const    new_symbol_buf,
                                esi_t          new_symbol_esi);

/**
* Submit a received repair symbol and try to progress in the decoding.
* For each decoded source symbol (if any), the application is informed
* through the dedicated callback functions.
*
* This function requires that the application has previously initialized
* the coding window and coding coefficients appropriately. The application
* keeps a full control of the repair symbol buffer, i.e., the application
* is in charge of freeing this buffer as soon as it believes appropriate
* (a copy is kept by the codec). This is motivated by the fact that a
* repair symbol may be part of a larger buffer (e.g., if there are
* several repair symbols per packet, or because of a packet header): only
* the application knows when the buffer can be safely freed.
*
* This function usually returns SWIF_STATUS_OK, regardless of whether
* this new symbol enabled the decoding of one or several source symbols,
* or SWIF_STATUS_ERROR. It cannot return SWIF_STATUS_FAILURE.
*
* @param dec    context (i.e., pointer to decoder structure).
* @param new_symbol_buf
*             (IN) Pointer to the new repair symbol now available (i.e.
*             a new symbol received by the application or a decoded
*             symbol in case of a recursive call if it makes sense).
* @return
*/
swif_status_t    swif_decoder_decode_with_new_repair_symbol (

```

```

                                swif_decoder_t* dec,
                                void* const      new_symbol_buf);
<CODE ENDS>

```

#### Decoder API proposal

```

<CODE BEGINS>
/**
 * Decoder structure that contains whatever is needed for decoding.
 * The exact content of this structure is FEC code dependent, the
 * structure below being a non normative example.
 * However it MUST be aligned with swif_decoder_t (same first items) in
 * order to be able to cast a pointer to one of the two structures,
 * depending on the context.
 */
typedef struct swif_decoder_internal {
    /* generic part of any control block. MUST be first in structure */
    swif_decoder_t  gen;

    /* desired verbosity: 0 is the minimum verbosity, the maximum
     * level being implementation specific. */
    uint32_t        verbosity;

    /* maximum number of source symbols used for any repair symbol */
    uint32_t        max_coding_window_size;

    /* max. number of source symbols kepts in current linear system.
     * If the linear system grows above this limit, old source
     * symbols in excess are removed and the application callback
     * called. This value should be larger than the
     * max_coding_window_size. */
    uint32_t        max_linear_system_size;

    /* exact size (in bytes) of any source or repair symbol */
    uint32_t        symbol_size;

    /* add whatever may be needed hereafter... */
} swif_decoder_internal_t;

```

Non normative example (RLC) of internal structure used by a decoder.

#### 4.4. Coding Window Functions at an Encoder and Decoder

This section gathers functions used to manage the coding window, both at an encoder and at a decoder. At an encoder a sliding (of fixed or elastic size) encoding window is managed. Whenever a repair symbol needs to be created, a linear combination (that is code specific) of source symbols currently in the encoding window is performed. This

encoding window is managed with the functions below plus, potentially, internal mechanisms that are code specific.

At a decoder, before submitting a new repair symbol to the codec, the application must specify the associated encoding window used at the source. This is done by the reset/add a single or set of symbols/remove a symbol functions. Once this coding window is ready, as well as the coding coefficient list if applicable, the application calls the `decode_with_new_repair_symbol()` function. A coding window may be reused for several repair symbols as long as they are all built from the same set of source symbols. In that case resetting the coding window and setting it from scratch would be a waste of time. The coding window must be viewed as a temporary list used solely by the `decode_with_new_repair_symbol()` function and kept independent from the linear system managed by the codec.

<CODE BEGINS>

```
/**
 * This function resets the current coding window. We assume here that
 * this window is maintained by the FEC codec instance.
 * Encoder:      reset the encoding window for the encoding of future
 *               repair symbols.
 * Decoder:      reset the coding window under preparation associated to
 *               a repair symbol just received.
 *
 * @return
 */
swif_status_t    swif_encoder_reset_coding_window (swif_encoder_t*  enc);

swif_status_t    swif_decoder_reset_coding_window (swif_decoder_t*  dec);

/**
 * Add this source symbol to the coding window.
 * Encoder:      add a source symbol to the coding window.
 * Decoder:      add a source symbol to the coding window under preparation.
 *
 * @param new_src_symbol_buf    (encoder only) pointer to a buffer
 *                               containing the source symbol. The application MUST NOT
 *                               free nor modify this buffer as long as the source symbol
 *                               is in the coding window.
 * @param new_src_symbol_esl    ESI of the source symbol to add.
 * @return
 */
swif_status_t    swif_encoder_add_source_symbol_to_coding_window (
                                swif_encoder_t*  enc,
                                void*            new_src_symbol_buf,
                                esi_t            new_src_symbol_esl);
```

```

swif_status_t    swif_decoder_add_source_symbol_to_coding_window (
                                swif_decoder_t* dec,
                                esi_t            new_src_symbol_esi);

/**
 * Remove this source symbol from the coding window.
 *
 * Encoder:      remove a source symbol from the encoding window, e.g.
 *                because the application knows that a source symbol has
 *                been acknowledged by the peer (if applicable). Note that
 *                the left side of the sliding window is automatically
 *                managed by the codec and no action is needed from the
 *                application. If needed a callback is available to inform
 *                the application that a source symbol has been removed).
 * Decoder:      remove a source symbol from the coding window under
 *                preparation.
 *
 * @param old_src_symbol_esi    ESI of the source symbol to remove from
 *                               the coding window.
 * @return
 */
swif_status_t    swif_encoder_remove_source_symbol_from_coding_window (
                                swif_encoder_t* enc,
                                esi_t            old_src_symbol_esi);

swif_status_t    swif_decoder_remove_source_symbol_from_coding_window (
                                swif_decoder_t* dec,
                                esi_t            old_src_symbol_esi);

<CODE ENDS>

```

Coding Window Functions at an Encoder and Decoder.

#### 4.5. Coding Coefficients Functions at an Encoder and Decoder

This section gathers functions used to manage the coding coefficients, both at an encoder and at a decoder. Since different FEC codecs will have different requirements, it is important to keep these functions separate from the `build_repair_symbol()` and `decode_with_new_repair_symbol()` functions. Several situations exist:

- o the application provides the list of coding coefficients to use for the next `build_repair_symbol()`;
- o the application provides a key (typically a PRNG seed) that the codec uses to produce the coding coefficients to use for the next `build_repair_symbol()`;
- o the choice of the coding coefficients is totally performed by the codec, in an autonomous manner (e.g., the codec includes an

algorithm that produces an appropriate seed based on various criteria, or the codec selects a set of coding coefficients based on various criteria). In that case the application needs to retrieve the list of coding coefficients or the key selected by the codec;

<CODE BEGINS>

```
/**
 * The following functions enable an encoder (resp. decoder) to
 * initialize the set of coefficients to be used for encoding
 * or associated to a received repair symbol.
 *
 * Encoder: calling one of them MUST be done before calling
 *          build_repair_symbol().
 * Decoder: calling one of them MUST be done before calling
 *          decode_with_new_repair_symbol().
 */

/**
 * Encoder: this function specifies the coding coefficients chosen by
 * the application if this is the way the codec works.
 * Decoder: communicate with this function the coding coefficients
 * associated to a repair symbol and carried in the packet
 * header.
 *
 * @param coding_coefs_tab
 * (IN) table of coding coefficients associated to each of
 * the source symbols currently in the encoding window.
 * The size (number of bits) of each coefficient depends on
 * the FEC Scheme. The allocation and release of this table
 * is under the responsibility of the application.
 * @param nb_coefs_in_tab
 * (IN) number of entries (i.e., coefficients) in the table.
 * @return
 */
swif_status_t swif_encoder_set_coding_coefs_tab (
                                swif_encoder_t* enc,
                                void*          coding_coefs_tab,
                                uint32_t        nb_coefs_in_tab);

swif_status_t swif_decoder_set_coding_coefs_tab (
                                swif_decoder_t* dec,
                                void*          coding_coefs_tab,
                                uint32_t        nb_coefs_in_tab);

/**
 * The coding coefficients may be generated in a deterministic manner,
```

```
* for instance by a PRNG known by the codec and a seed (perhaps with
* other parameters) provided by the application.
* The codec may also choose in an autonomous manner these coefficients.
* This function is used to trigger this process.
* When the choice is made in an autonomous manner, the actual coding
* coefficient or key used by the codec can be retrieved with
* swif_encoder_get_coding_coefs_tab().
*
* @param key      (IN) Value that can be used as a seed in case of a PRNG
*                  for instance, or by a specific coding coefficients
*                  function. Set to 0 if not required by a codec.
* @param add_param
*                  (IN) an opaque 32-bit integer that contains a codec
*                  specific parameter if needed. Set to 0 if not used.
* @return
*/
swif_status_t    swif_encoder_generate_coding_coefs (
                                swif_encoder_t* enc,
                                uint32_t        key,
                                uint32_t        add_param);

swif_status_t    swif_decoder_generate_coding_coefs (
                                swif_decoder_t* dec,
                                uint32_t        key,
                                uint32_t        add_param);

/**
* This function enables the application to retrieve the set of coding
* coefficients generated and used by build_repair_symbol(). This is
* useful when the choice of coefficients is performed by the codec in
* an autonomous manner but needs to be sent in the repair packet header.
* This function is only used by an encoder.
*
* @param coding_coefs_tab
*        (OUT) pointer to a table of coding coefficients.
*        The size (number of bits) of each coefficient depends on
*        the FEC scheme. Upon return of this function, this table
*        is allocated and filled with coefficient values. The
*        release of this table is under the responsibility of the
*        application.
* @param nb_coefs_in_tab
*        (IN/OUT) pointer to the number of entries (i.e.,
*        coefficients) in the table.
*        Upon calling this function, this number must be zero.
*        Upon return of this function this variable is initialized
*        with the actual number of entries in the coeffs_tab[].
* @return
```



```

*/
swif_status_t    swif_encoder_get_coding_coefs_tab (
                                swif_encoder_t* enc,
                                void**          coding_coefs_tab,
                                uint32_t*       nb_coefs_in_tab);

/**
 * Get information on the current coding window at the encoder.
 * This function stores the ESI of the first source symbol and
 * last source symbol in the coding window, as well as the number
 * of symbols. In theory the application should be able to recover
 * the information (it knows when new symbols are added and old
 * symbols removed), but it's easier to let the SWiF Codec care
 * about it. The number of source symbols is also returned.
 * In situations where there's no gap (i.e., when
 * swif_encoder_remove_source_symbol_from_coding_window() has not
 * been used), nss can also be calculated with first/last. However
 * it is more convenient to use nss directly (in particular in case
 * of wrapping to zero of either first or last).
 *
 * @param enc
 * @param first      (in/out) pointer to ESI of the first source
 *                   symbol in the coding window (inclusive)
 * @param last       (in/out) pointer to ESI of the last source
 *                   symbol in the coding window (inclusive)
 * @param nss        (in/out) pointer to number of source symbols
 *                   in the coding window
 * @return
 */
swif_status_t    swif_encoder_get_coding_window_information (
                                swif_encoder_t* enc,
                                esi_t*         first,
                                esi_t*         last,
                                uint32_t*      nss);
<CODE ENDS>

```

Coding Coefficients Functions at an Encoder and Decoder.

## 5. Security Considerations

TBD

## 6. IANA Considerations

This document has no IANA requirement.

## 7. Acknowledgments

The authors would like to thank Marie-Jose Montpetit, Francois Michel, and Oumaima Attia for their valuable contributions to the IETF Hackathon SWiF-Codec project and their inputs to this document.

## 8. References

### 8.1. 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>>.

### 8.2. Informative References

- [fecframe-ext] Roca, V. and A. Begen, "Forward Error Correction (FEC) Framework Extension to Sliding Window Codes", Transport Area Working Group (TSVWG) draft-ietf-tsvwg-fecframe-ext (Work in Progress), June 2018, <<https://tools.ietf.org/html/draft-ietf-tsvwg-fecframe-ext>>.
- [RLC] Roca, V. and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Scheme for FECFRAME", Transport Area Working Group (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in Progress), June 2018, <<https://tools.ietf.org/html/draft-ietf-tsvwg-rlc-fec-scheme>>.

## Authors' Addresses

Vincent Roca  
INRIA  
Univ. Grenoble Alpes  
France

EMail: [vincent.roca@inria.fr](mailto:vincent.roca@inria.fr)

Jonathan Detchart  
ISAE - Supaero  
France

EMail: [jonathan.detchart@isae-supaero.fr](mailto:jonathan.detchart@isae-supaero.fr)

Cedric Adjih  
INRIA  
France

EMail: [cedric.adjih@inria.fr](mailto:cedric.adjih@inria.fr)

Morten V. Pedersen  
Steinwurf ApS  
Denmark

EMail: [morten@steinwurf.com](mailto:morten@steinwurf.com)