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.
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

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| T |SYMBOLS| ENCODER RANK | SEED or CODING COEFFICIENTS |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

SYMBOL(S) DATA

...
2.2. Field Types and Formats

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

- 2 bits long.
- 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.
- 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.
- 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:

- 4 bits long. A maximum number of 15 symbols are concatenated within each symbol representation.
- 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 cases 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:

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

- The SEED field is only present when TYPE flag is '2'. If TYPE is '1' or '3', this field is absent.
- 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.
- 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.
- 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 \(|SEED| \times |SYMBOLS| \times |ENCODER RANK| \) which in the best case is \((2^8)\times(2^4-1)\times(2^{10}) = 2^{22}\), 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.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
| 1 | 3 | 0 | Systematic Symbol 0 Data |
+-----------------------------------------------+
| Systematic Symbol 1 Data | Systematic Symbol 2 Data |
```

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.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
| 2 | 2 | 8 | 4 | Coded Symbol A |
+-----------------------------------------------+
```

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.
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 COE芙FS).

![Figure 8: A systematic symbol representation.](image-url)

![Figure 7: A symbol representation with 2 recoded symbols having coding coefficients attached.](image-url)
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

5.2. Informative References


Authors' Addresses

Janus Heide
Steinwurf Aps
Aalborg
Denmark

Email: janus@steinwurf.com

Shirley Shi
Code On Network Coding LLC
Cambridge
USA

Email: xshi@alum.mit.edu

Kerim Fouli
Code On Network Coding LLC
Cambridge
USA

Email: fouli@codeontechnologies.com

Muriel Medard
Code On Network Coding LLC
Cambridge
USA

Email: muriel.medard@codeontechnologies.com
Vince Chook
Inmarsat PLC
London
United Kingdom

Email: Vince.Chook@inmarsat.com
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."
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.
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.

![Diagram of sliding-window RLNC](image)

**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:

- 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]

- 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.

- 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.

- 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

```
+-----------+---------+---------+
| C |    D1   |    D2   |
+-----------+---------+---------+    +---->
```

(b) Conventional fragmentation

```
+-----------+---------+---------+
|    D1     |    D2     |   +---->
+-----------+---------+---------+
```

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.
2.1.2. 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


5.2. Informative References


Authors' Addresses

Janus Heide
Steinwurf Aps
Aalborg
Denmark

Email: janus@steinwurf.com

Shirley Shi
Code On Network Coding LLC
Cambridge
USA

Email: xshi@alum.mit.edu

Kerim Fouli
Code On Network Coding LLC
Cambridge
USA

Email: fouli@codeontechnologies.com
Muriel Medard
Code On Network Coding LLC
Cambridge
USA

Email: muriel.medard@codeontechologies.com

Vince Chook
Inmarsat PLC
London
United Kingdom

Email: Vince.Chook@inmarsat.com
Coding techniques for satellite systems
draft-irtf-nwcrg-network-coding-satellites-06

Abstract

This document is the product of the Coding for Efficient Network Communications Research Group (NWCRG). This document follows the taxonomy document [RFC8406] and considers coding as a linear combination of packets that operate in and above the network layer. In this context, this memo details a multi-gateway satellite system to identify use-cases where coding is relevant. As example, coding operating in and above the network layer can be exploited to cope with residual losses or provide reliable multicast services. The objective is to contribute to a larger deployment of such techniques in SATCOM systems. This memo also identifies open research issues related to the deployment of coding in SATCOM systems, such as the interaction between congestion controls and coding techniques.

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 February 20, 2020.

Copyright Notice

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

Guaranteeing both physical-layer robustness and efficient usage of the radio resource has been in the core design of SATellite COMMunication (SATCOM) systems. The trade-off often resided in how much redundancy a system adds to cope with link impairments, without reducing the goodput when the channel quality is good. There is usually enough redundancy to guarantee a Quasi-Error Free transmission. The recovery time depends on the encoding block size. Considering for instance geostationary satellite system (GEO), physical or link layer erasure coding mechanisms recover transmission losses within a negligible delay compared to link delay. However,
when retransmissions are triggered, this leads to a non-negligible additional delay in particular over GEO link. Further exploiting coding techniques at application or transport layers is an opportunity for releasing constraints on the physical layer and improving the performance of SATCOM systems.

The notations used in this document are based on the taxonomy document [RFC8406]:

- Channel and link codings are gathered in the PHY layer coding and are out of the scope of this document.
- FEC (also called Application-Level FEC) operates in and above the network layer.
- This document considers coding (or coding techniques or coding schemes) as a linear combination and not as a content coding (e.g., to compress a video flow).

Figure 1 presents the status of the reliability schemes deployment in satellite systems.

- X1 embodies the source coding techniques that could be used at application level for instance within QUIC. This is not specific to SATCOM systems since such deployment can be relevant for broadband Internet access discussions.
- X2 embodies the physical-layer techniques exploited in SATCOM systems (note that other coding techniques can be exploited). This is out of the scope of the document.

<table>
<thead>
<tr>
<th></th>
<th>Upper Appl.</th>
<th>Middle ware</th>
<th>Communication layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source coding</td>
<td>X1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network AL-FEC</td>
<td></td>
<td>X1</td>
<td></td>
</tr>
<tr>
<td>Packetization UDP/IP</td>
<td></td>
<td>X2</td>
<td></td>
</tr>
<tr>
<td>PHY layer</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Reliability schemes in current satellite systems
We notice an active research activity on coding techniques and SATCOM. That being said, not much has actually made it to industrial developments. In this context, this document aims at identifying opportunities for further usage of coding in these systems.

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

- ACM: Adaptive Coding and Modulation;
- 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;
- CPE: Customer Premises Equipment;
- COM: COMmunication;
- DSL: Digital Subscriber Line;
- DTN: Delay/Disruption Tolerant Network;
- DVB: Digital Video Broadcasting;
- E2E: End-to-end;
- ETSI: European Telecommunications Standards Institute;
- FEC: Forward Error Correction;
- FLUTE: File Delivery over Unidirectional Transport;
- IntraF: Intra-Flow Coding;
- InterF: Inter-Flow Coding;
- IoT: Internet of Things;
- LTE: Long Term Evolution;
- MPC: Multi-Path Coding;
- NC: Network Coding;
- NFV: Network Function Virtualization;
This document is the product of and represents the collaborative work and consensus of the Coding for Efficient Network Communications Research Group (NWCRG); it is not an IETF product and is not a standard.

2. A note on satellite topology

This section describes a satellite system that follows the ETSI DVB standards to provide broadband Internet access. A high-level description of a multi-gateway satellite network is provided. There are multiple SATCOM systems, such as those dedicated to broadcasting TV or to IoT applications: depending on the purpose of the SATCOM system, ground segments are specific. In this context, the increase of the available capacity that is carried out to end users and reliability requirements lead to multiple gateways for one unique satellite platform.

In this context, Figure 2 shows an example of a multi-gateway satellite system. In a multi-gateway system, some elements may be centralized and/or gathered: the relevance of one approach compared to another depends on the deployment scenario. More information on these discussions and a generic SATCOM ground segment architecture for bidirectional Internet access can be found in [SAT2017].
Some functional blocks aggregate the traffic of multiple users.
Figure 2: Data plane functions in a generic satellite multi-gateway system
3. Use-cases for improving the SATCOM system performance with coding techniques

This section details use-cases where coding techniques could provide interesting features for SATCOM systems. Combination of the presented use-cases could also be relevant.

It is worth noting that these use-cases mostly focus on the middleware and packetization UDP/IP of Figure 1. There are already lots of recovery mechanisms at the physical-layer in currently deployed systems while E2E source coding is done at the application level. In a multi-gateway SATCOM Internet access, the deployment opportunities are more relevant in specific SATCOM components such as the "network function" block or the "access gateway" of Figure 2.

3.1. Two-way relay channel mode

This use-case considers a two-way communication between end users, through a satellite link. Figure 3 proposes an illustration of this scenario.

Satellite terminal A sends a flow A and satellite terminal B sends a flow B to a coding server. The coding server sends a combination of both terminal flows. This results in non-negligible capacity savings and has been demonstrated [ASMS2010]. In the proposed example, a dedicated coding server is introduced. Its location could be changed depending on the deployment use-case. With On-Board Processing satellite payloads, the coding operations could be done at the satellite level; although this would require lots of computational resource on-board and may not be relevant with today’s payloads.

\[
\begin{align*}
-X\} & : \text{traffic from satellite terminal X to the server} \\
={X+Y=} & : \text{traffic from X and Y combined sent from} \\
& \text{the server to terminals X and Y}
\end{align*}
\]

Figure 3: Network architecture for two way relay channel with NC
3.2. Reliable multicast

Using multicast servers is a way to better exploit the satellite broadcast capabilities. This approach 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-negligible scarce resource saving. We propose an illustration for this scenario in Figure 4.

- \(-\text{Li}\) : packet indicating the loss of packet \(i\) of a multicast flow \(M\)
- \(\text{=\{}M\text{=}\} :\) multicast flow including the missing packets

\begin{verbatim}
+-----------+       +-----+
|Sat term A |  -Li}-+ |     |
|-----------|     |     |      +---------+  +------+
|^|          | SAT | -Li}--| Gateway |  Multi |
|==={M=====|     |={M==|         |  |Server|
|vv         |     |      +---------+  +------+
+-----------+     |     |
|Sat term B |  -Lj}-+ |     |
|-----------+       +-----+
\end{verbatim}

Figure 4: Network architecture for a reliable multicast with NC

A multicast flow \((M)\) is forwarded to both satellite terminals A and B. However packet \(N_i\) (resp. \(N_j\)) gets lost at terminal A (resp. B), and terminal A (resp. B) returns a negative acknowledgment \(L_i\) (resp. \(L_j\)), indicating that the packet is missing. Then either the access gateway or the multicast server includes a repair packet (rather than the individual \(N_i\) and \(N_j\) packets) in the multicast flow to let both terminals recover from losses.

This could be achieved by using other multicast or broadcast systems, such as NACK-Oriented Reliable Multicast (NORM) [RFC5740] or File Delivery over Unidirectional Transport (FLUTE) [RFC6726]. Note that both NORM and FLUTE are limited to block coding, none of them supporting sliding window encoding schemes [RFC8406]. Note that although FLUTE is defined as an unidirectional protocol, the RFC proposes a bidirectional communication method to enable full reliability transfer and for security purposes.
3.3. Hybrid access

This use-case considers the use of multiple path management with coding at the transport layer to increase the reliability and/or the total capacity (using multiple paths does not guarantee an improvement of both the reliability and the total capacity). We propose an illustration for this scenario in Figure 5. This use-case is inspired from the Broadband Access via Integrated Terrestrial Satellite Systems (BATS) project and has been published as an ETSI Technical Report [ETSITR2017]. This kind of architecture is also discussed in the TCPM working group [I-D.ietf-tcpm-converters].

To cope with packet loss (due to either end-user mobility or physical-layer impairments), coding techniques could be introduced both at the CPE and at the concentrator. Apart from packet losses, other gains could be envisioned, such as a better tolerance to out-of-order packets which occur when exploited links exhibit high asymmetry in terms of RTT. Depending on the ground architecture [I-D.chin-nfvrg-cloud-5g-core-structure-yang] [SAT2017], some equipments might be hosting both SATCOM and cellular functions.

-{}- : bidirectional link

-{}- SAT |{}- BACKBONE
--------- +{}- CPE |{}- CONCENTRATOR

--- DSL |{}- Server

-{}- LTE

Figure 5: Network architecture for an hybrid access using coding

3.4. Dealing with LAN losses

This use-case considers the usage of coding techniques to cope with cases where the end user connects to the satellite terminal with a Wi-Fi link that exhibits losses. In the case of encrypted end-to-end applications based on UDP, PEP cannot operate. The Wi-Fi losses result in an end-to-end retransmission that would harm the quality of experience of the end user.

The architecture is recalled in Figure 6.
In this use-case, adding coding techniques could prevent the end-to-end retransmission from occurring.

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

Figure 6: Network architecture for dealing with LAN losses

3.5. Dealing with varying channel conditions

This use-case considers the usage of coding techniques to cope with cases where channel condition can change in less than a second and where the physical-layer codes could not efficiently guarantee a Quasi-Error-Free (QEF) transmission.

The architecture is recalled in Figure 7. In these cases, the mechanisms that are exploited to adapt the physical-layer codes (Adaptive Coding and Modulation (ACM)) may adapt the modulation and coding in time: remaining errors could be recovered with higher layer redundancy packets. Coding may be applied on IP packets or on layer-2 proprietary format packets.

This use-case is mostly relevant when mobile users are considered or when the chosen band induces a required physical-layer coding that may change over time (Q/V bands, Ka band, etc.). Depending on the use-case (e.g., very high frequency bands, mobile users) or depending on the deployment use-cases (e.g., performance of the network between each individual block), the relevance of adding coding techniques is different.

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

Figure 7: Network architecture for dealing with varying link characteristics
3.6. Improving the gateway handovers

This use-case considers the recovery of packets that may be lost during gateway handovers. Whether this is for off-loading one given equipment or because the transmission quality is not the same on each gateway, changing the transmission gateway may be relevant. However, if gateways are not properly synchronized or if the algorithm that is exploited to trigger gateway handovers shows a non negligible probability of missed detection, this may result in packet losses. During these critical phases, coding can be added to improve the reliability of the transmission and allow a seamless gateway handover. Coding could be applied at either the access gateway or the network function block. A potential control plane is in charge of taking the decision to change the communication gateway and the consequent routes.

Figure 8 illustrates this use-case.

-{}- : bidirectional link
! : management interface
C : where coding techniques could be introduced

Figure 8: Network architecture for dealing with gateway handover schemes

4. Research challenges

This section proposes a few potential approaches to introduce and use coding techniques in SATCOM systems.
4.1. On the joint-use of coding techniques and congestion control in SATCOM systems

SATCOM systems typically feature Performance Enhancement Proxy (PEP) RFC 3135 [RFC3135]. PEP usually split TCP end-to-end connections and forward TCP packets to the satellite baseband gateway that deals with layer-2 and layer-1 encapsulations. PEP contributes to mitigate congestion in a SATCOM systems. PEP could host coding mechanisms and thus support use-cases that have been discussed in this document.

Deploying coding schemes at the TCP level in these equipment could be relevant and independent from the specific characteristics of a SATCOM link. This leads to research questions on the interaction between coding schemes and TCP congestion controls.

4.2. On the efficient usage of satellite resource

The recurrent trade-off in SATCOM systems remains: 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 coding could be added?

This problem has been tackled in the past for physical-layer code, but there remains questions on how to adapt the overhead for, e.g., the quickly varying channel conditions use-case.

4.3. Interaction with virtualized satellite gateways and terminals

Related to the foreseen virtualized network infrastructure, coding techniques could be easily deployed as VNF. Next generation of SATCOM ground segments could rely on a virtualized environment. This trend can also be seen in cellular networks, making these discussions extendable to other deployment scenarios [I-D.chin-nfvrg-cloud-5g-core-structure-yang]. As one example, the coding VNF deployment in a virtualized environment is 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, to guarantee efficient radio usage and easy-to-deploy SATCOM services. Moreover, another challenge related to a virtualized SATCOM equipment is the management of limited buffered capacities.
4.4. Delay/Disruption Tolerant Networks

Communications among deep-space platforms and terrestrial gateways can be a challenge. Reliable end-to-end (E2E) communications over such paths must cope with long delay and frequent link disruptions; indeed, contemporaneous E2E connectivity may be available only intermittently or never. Delay/Disruption Tolerant Networking [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 the data between a central PEP and a remote PEP.

Coding enables E2E reliable communication over DTN with adaptive re-encoding, as proposed in [THAI15]. In this case, the use-cases proposed in Section 3.5 would legitimize the usage of coding within the DTN stack to improve the channel utilization and the E2E transmission delay. In this context, the use of erasure coding techniques inside a Consultative Committee for Space Data Systems (CCSDS) architecture has been specified in [CCSDS-131.5-O-1]. A research challenge would be on how such coding can be integrated in the IETF DTN stack.

5. Conclusion

This document discusses some opportunities to introduce coding techniques at a wider scale in satellite telecommunications systems.

Even though this document focuses on satellite systems, it is worth pointing out that some scenarios proposed may be relevant to other wireless telecommunication systems. As one example, the generic architecture proposed in Figure 2 may be mapped to 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 draft since they may be also relevant for cellular networks.

6. 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.
7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Security considerations are inherent to any access network, and in particular SATCOM systems. The use of FEC or Network Coding in SATCOM also 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). However this is not specific to the SATCOM use-case and this document does not further elaborate on it.

9. Informative References

[ASMS2010]

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

[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.ietf-tcpm-converters]

[I-D.vazquez-nfvrg-netcod-function-virtualization]


Authors’ Addresses

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

Email: nicolas.kuhn@cnes.fr

Emmanuel Lochin (editor)
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse 31400
France

Email: emmanuel.lochin@isae-supaero.fr
Network Coding for Content-Centric Networking / Named Data Networking: Requirements and Challenges
draft-irtf-nwcrg-nwc-ccn-reqs-02

Abstract

This document describes the current research outcomes regarding Network Coding (NC) for Content-Centric Networking (CCN) / Named Data Networking (NDN), and clarifies the requirements and challenges for applying NC into CCN/NDN.

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 23, 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
1. Introduction

Information-Centric Networks (ICN) in general, and Content-Centric Networking (CCN) [15] or Named Data Networking (NDN) [17] in particular, have emerged as a novel communication paradigm advocating to retrieve data through 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 CCN/NDN architecture has introduced innovative ideas and has stimulated research in a variety of areas, such as in-network caching, name-based routing, multi-path transport, content security, and so on. One key benefit of requesting content by name is that it removes the need to establish a session between the client and a specific server, and that content can thereby be retrieved from multiple sources.
In parallel, there has been a growing interest from both academia and industry to better understand 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, point-to-multipoint connections, etc. Typically, NC is a technique mainly used to encode packets to recover lost source packets at the receiver, and to effectively get the desired information in a fully distributed manner. In addition, NC can be used for security enhancements [2] [3] [4] [5].

From the perspective of NC transport mechanism, NC is divided into two major categories: one is coherent NC, and the other is non-coherent NC [33]. In coherent NC, source and destination nodes exactly know network topology and coding operations at intermediate nodes. When multiple consumers are trying to receive the same content such as live video streaming, coherent NC could enable the optimal throughput by making the content flow sent over the constructed optimal multicast trees [26]. However, it requires a fully adjustable and specific routing mechanism, and an intense computational task for central coordination. In the case of non-coherent NC that often utilizes RLC, it is not required to know either network topology nor intermediate coding operations [27]. Since non-coherent NC works in a completely independent and decentralized manner, this approach is more feasible especially in the large scale use cases.

NC mixes multiple packets together with parts of the same content, and may do this at the source or at other nodes in the network. As such, network coded packets are not connected to a specific server, as they may have been mixed within the network. Since NC focuses on what information should be encoded in a network packet, rather than the specific host where it has been generated, it is in line with the CCN/NDN core networking layer. NC has already been implemented for information/content dissemination (e.g. [6] [7] [8]). Montpetit, et al., first suggested to exploit NC techniques to enhance key system performances in ICN [9]. NC provides CCN/NDN with the highly beneficial potential to effectively disseminate information in a completely independent and decentralized manner.

In this document, we consider how NC can be applied to the CCN/NDN architecture and describe the requirements and potential challenges for making CCN/NDN-based communications better using the NC technology. Please note that providing specific solutions (e.g., NC optimization methods) to enhance CCN/NDN performance metrics by exploiting NC is out of scope of this document.
2. Terminology

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 [1].

2.1. Definitions

The terminology regarding NC used in this document is described below. It is aligned with RFCs produced by the FEC Framework (FECFRAME) IETF Working Groups as well as IRTF Coding for Efficient Network Communications Research Group (NWCRG) [20].

- Random Linear Coding (RLC): Particular case of Linear Coding using a set of random coding coefficients.
- Generation, or (IETF) Block: With Block Codes, the set of Source Symbols of the input Flow(s) that are logically grouped into a Block, before doing encoding.
- Generation Size, Code Dimension, or (IETF) Block Size: With Block Codes, the number of Source Symbols, k, belonging to a Block.
- Coding Vector: A set of Coding Coefficients used to generate a certain Repair Symbol through Linear Coding. The number of nonzero coefficients in the Coding Vector defines its density.
- Finite Field: Finite fields, used in Linear Codes, have the desired property of having all elements (except zero) invertible for + and * and all operations over any elements do not 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.
- Finite Field size: The number of elements in a finite field. For example the binary extension field {0..2^m-1} has size q=2^m.
- Block Coding: Coding technique where the input Flow(s) must be first segmented into a sequence of blocks, FEC encoding and decoding being performed independently on a per-block basis.
- Sliding Window Coding or Convolutional Coding: General class of coding techniques that rely on a sliding encoding window. This is an alternative solution to Block Coding.
- Fixed or Elastic Sliding Window Coding: Coding technique that generates repair data on-the-fly, from the set of source data.
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").

- Feedback: Feedback information sent by a decoding node to a node (or from a consumer to a publisher in case of End-to-End Coding). The nature of information contained in a feedback packet varies, depending on the use-case. It can provide reception and/or decoding statistics, or the list of available source packets received or decoded, or the list of lost source packets that should be retransmitted, or a number of additional repair packet needed to have a full rank linear system.

Concerning CCN/NDN, the following terminology and definitions are used.

- Consumer: A node requesting content. It initiates communication by sending an interest packets.

- Publisher: A node providing content. It originally creates or owns the content.

- Forwarding Information Base (FIB): A lookup table in a content router containing the name prefix and corresponding destination interface to forward the interest packets.

- Pending Interest Table (PIT): A lookup table populated by the interest packets containing the name prefix of the requested data, and the outgoing interface used to forward the received data packets.

- Content Store (CS): A storage space for a router to cache content objects. It is also known as in-network cache.

- Content Object: A unit of content data delivered through the CCN/NDN network.

- Content Flow: A sequence of content objects associated with the unique content name prefix.

2.2. NDN/CCN Background

Armed with the terminology above, we briefly explain the key concepts of CCN/NDN. Both protocols are similar in principle, and different on some implementation choices.
In a CCN network, there are two types of packets at the network level: interest and data. The consumer requests a content by sending an "interest" message, that carries the name of the data. On difference to note here in CCN and NDN is that in CCN [16], the interest must carry a full name, while in NDN [18] it may carry a name prefix (and receive in return any data with a name matching this prefix).

Once a router receives an "interest" message, it performs a series of look-up: first it checks in the Content Store if it has a copy of the requested content available. If it does, it returns the data and the transaction has successfully completed.

If it does not, it performs a look-up of the PIT to see if there is already an outgoing request for the same data. If there is not, 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 used later to send the data back, since interest packets do not carry a source field that identifies the requester. If there is already a PIT entry for this name, then it is updated with the incoming interface of this new request and the interest is discarded.

After the PIT look-up, the interest undergoes a FIB lookup to select an outgoing interface. The FIB lists name prefixes and their corresponding forwarding interfaces, to send the interface towards a router that possesses a copy of the requested data.

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

Data packets carry some information to validate the data, in particular that the data is indeed the one that corresponds to the name. This is required since authentication of the object is crucial in CCN/NDN. However, this step is optional at intermediate routers, so as to speed up the processing.

The key aspect of CCN/NDN is that the consumer of the content does not establish a session with a specific server. Indeed, the node that returns the content is not aware of the network location of the requester and the requester 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 totally different networks.
3. Advantages provided by NC and CCN/NDN

Both NC for large scale content dissemination [7] and CCN/NDN can contribute to effective content/information delivery while working jointly. They both bring 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 to combine [19], while the latter focuses on content/information itself at the networking layer. Because these approaches are complementary, it is natural to combine them.

The CCN/NDN core abstraction at networking layer through name makes network stack simple as it enables applications to take maximum advantage of multiple simultaneous connectivities due to its simpler relationship with the layer 2 [15]. CCN/NDN itself, however, does not provide reliable and robust content dissemination by default. This requires some specific CCN/NDN transport (i.e., strategy layer) [15]. NC can enable the CCN/NDN transport system to effectively distribute and cache data associated with multi-path data retrieval [9]. Furthermore, NC can contribute to improving both caching performance and cache privacy that CCN/NDN newly poses at the networking layer [25]. Others also have considered NC in CCN/NDN use cases such as content dissemination with in-network caching [10] [12] [13], seamless mobility [11] [31], low-latency video streaming [14], etc. In this context, it should be natural that there is much room for considering NC integration into CCN/NDN.

4. Requirements

This section presents the NC requirements for ICN/CCN in terms of network architecture and protocol. The current document focuses on NC in a block coding manner.

4.1. Content Naming

Naming content objects is as important for CCN/NDN as naming hosts is for today’s Internet [21]. Before performing network coding for specified content in CCN/NDN, the overall content should be split into small content objects to avoid packet fragmentation that could cause unnecessary packet processing and degrade throughput. The size of content objects should be within the allowable packet size so as to avoid packet fragmentation in CCN/NDN network, and then network coding should be applied into a set of the content objects.

Each coded packet MAY have a unique name as the original content object has in CCN/NDN, since PIT/CS operations typically need a unique name to identify the coded data. As a way of naming coded packet, the coding vector and the identifier of generation can be
used as a part of the content object name [10]. For instance, when the block size (also called generation size) is \( k \) and the coding vector is \( [1,0,0,0] \), the name would be like /CCN.com/video-A/k/1000. This naming scheme is simple and can support the delivery of coded packets with exactly the same operations in the PIT/CS as for original source packets. Since such a naming way enables consumer to specify coded packets to receive, it could shift the generation of the coding vector from the content producer onto the content requester (described in Section 4.2.2).

If a naming schema such as above is used, it would be valuable to reconsider whether Interests should carry full names (as in CCN) or prefixes (as in NDN) as multiple network coded packets could match a response to a specific prefix for a given generation, such as /CCN.com/video-A/k. In the latter case allowing partial name matching, the content requester may not be able to obtain degrees of freedom. Thus, extensions in the TLV header of the Interest would be used to specify further network coding information so as to limit coded packets to be received (for instance, by specifying the encoded vectors the content requester receives (also called decoding matrix) as in [9]). However, it may incur a largely increased size of TLV header, and thus it may be useful to use compression techniques for coding vectors [22] [23]. Without such coding information, the forwarding node would need to maintain some records regarding interest packets sent before (described in Section 4.2.3).

Coded packet MAY have a name that indicates that it is a coded packet, and move the coding information into a metadata field in the payload (i.e., the name includes only data type, original or coded packet, etc). It would not be beneficial for applications or services that may not need to understand the packet payload. Due to the possibility that multiple coded packets may have a same name, some mechanism is needed for the content requester to obtain innovative coded packets. As described in Section 4.3, a mechanism to manage the multiple innovative packets in the CS would be required as well. In addition, extra computational overhead would occur when the payload is being encrypted (described in Section 4.5).

4.2. Transport

The pull-based request-response feature of CCN/NDN is a fundamental principle of its transport layer; one Interest retrieves at most one Data packet. It is believed that it is important to not violate this rule, as it would open denial of service attacks issues, and thus the following basic operation should be considered to apply NC to CCN/NDN. In any case, such security considerations must be addressed if this rule were to be violated.
4.2.1. Scope of Network Coding

It should be discussed whether the network can recode data packets that are being received in transit, or if only the data that matches an interest can be subject to network coding operations. In the latter case, the network coding is performed on an end-to-end basis (where one end is the consumer, and the other end is any node that is able to respond to the Interest). In the former case, NC happens anywhere in the network that is able to update the data. As CCN/NDN has mechanisms in place to ensure the integrity of the data during transfer, NC in the network introduce complexities that would require special consideration for the integrity mechanisms to still work.

Similarly, caching of network coded packets at intermediate node may be valuable, but may prevent the node caching the coded content to validate the content.

4.2.2. Consumer Operation

To obtain NC benefits associated with in-network caching, consumer needs to issue interests directing the router (or publisher) to forward innovative coded packets if available. The reason why this directive is needed is that delay-sensitive applications such as live-video streaming may want to sequentially get original packets rather than coded packets cached in routers due to real-time constraint. As described in Section 4.1, because coded packet can have a name explicitly different from original source packets, issuing such an interest is possible.

When issuing interests specifying unique names with k and coding vectors for each coded packets, consumer appropriately receives innovative coded packets if they are available at some nodes and can be forwarded to the consumer. However, consumer needs to know the naming structure (through a specific name resolution scheme for instance) in order for nodes to specify the exact name of generated coded data packet to retrieve it. In the case of NC end-to-end approach, if consumer want to adjust some coding parameters at publisher, some specific scheme would be required.

4.2.3. Router Operation

Routers need to forward linearly independent coded packets toward downstream nodes if incoming interests for coded packets does not specify some coding parameters such as the coding vector to be used. Routers thus need to determine whether or not they can generate useful coded packets for consumers. Assuming that the size of the Finite Field in use is not relatively small, re-encoding using enough cached independent packets has a strong probability of making
independent coded packets [26]. However, without enough cached packets, router needs to determine whether or not an independent coded packet can be forwarded to the interface at which the interest arrived. To deal with this issue, some proposed schemes [10] require that the router maintains a tally of the interests for a specific name, generation and the corresponding interface, so as to know how many degrees of freedom have been provided already for the NC packets. Scalability and practicality of maintaining such scheme at intermediate routers should be considered. In addition, some transport mechanism of in-network loss detection and recovery [31] [14] at router as well as consumer-driven mechanism could be indispensable in order to enable fast loss recovery and enhance NC gains. After determining that independent coded packet cannot be provided, according to the FIB, the router relays received interests to upstream nodes to receive a new original or independent coded packet. In this context, to effectively and quickly retrieve independent coded data, appropriately setting the FIB and efficient interest forwarding strategies should be also considered.

In another possible case, when receiving interests for only original packets, routers may try to decode and get all the original packets and store them (if there are fully available cache capacity), enabling faster response to the interests. Since re-encoding or decoding leads to extra computational overhead, routers need to determine how to respond to receiving interests according to the use case (e.g., delay-sensitive or delay-tolerant application) and the router situation such as available cache space and computational capability.

4.2.4. Publisher Operation

The procedure for splitting an overall content into small content objects (described in Section 4.1) is the responsibility of the original publisher. When applying NC for the content, the publisher performs NC over the content objects, and naming processing for the coded packets. If the producer takes the lead in determining the used coding vectors and generating the coded packets, there are the two possible end-to-end cases; 1) content requestors obtain the names of coded packets through a certain mechanism, and send the correspond interests toward the publisher to get the coded packets already generated at the publisher, and 2) the publisher determines the coding vectors after receiving interests specifying them. In the former case, although content requestors cannot flexibly specify an coding vector for generating the coded packet to retain, but the latency for getting the coded data can be reduced compared to the latter case where additional NC operations need after receiving interests. The common benefit in such end-to-end cases is that if the publisher adds signature on the coded packets, data verification
can be possible throughout. According to application requirement for latency, such NC operation strategy should be considered.

4.2.5. Backward Compatibility

Network Coding operations should be applied in addition to the regular network behavior. As such, nodes should be able to not support network coding (either in forwarding the packets, but also in the caching mechanism). Network Coding operations should function alongside regular network operations. A network coding framework should be compatible with a regular framework, so as to allow backward compatibility and smooth migration from one framework to the other.

4.3. In-network Caching

Caching is an essential technique to improve throughput and latency in various applications. In-network caching CCN/NDN essentially supports at network level is highly beneficial by exploiting NC to enable effective multicast transmission [32], multipath data retrieval [10] [11], fast loss recovery [14], and so on. However, there are several issues to be considered.

As a general issue, there are limitations of cache capacity, and caching policy affects on consumer’s performances [24] [28] [29]. It is thus highly significant for routers to determine which packets should be cached and discarded. Since delay-sensitive applications often do not require in-network cache for a long period due to their real-time constraints, routers have to know the necessity for caching received packets to save the caching volume. This could be possible by putting a flag into optional header of data packets at publisher side. When receiving data packets with the flag meaning no necessity for cache, routers just have to forward them to downstream nodes. On the other hand, when receiving original packets or coded packets without the flag, router may cache them based on a specified replacement policy.

One key aspect of in-network caching is whether or not intermediate nodes can cache NC packets without first decoding them. They may be caching the coded packets without having the ability to perform validation of the content (described in Section 4.5). Therefore, caching of coded packets would require some mechanism to validate coded packets. In addition, when coded packets have a same name, it would also require some mechanism to identify them.
4.4. Seamless Mobility

This subsection presents how NC can achieve seamless mobility [11] [31] and clarify the requirements. A key feature of CCN/NDN is that it is sessionless and that multiple interests can be sent to different copies of the content in parallel. CCN/NDN enables a consumer to retrieve the content from multiple copies that are distributed and asynchronous. The key benefit is that the link between the consumer and the multiple copies acts as a virtual logical link, upon which rate adaptation mechanism (say, for video streaming) can be performed.

In this context, NC adds a reliability layer network to CCN in a distributed and asynchronous manner, because NC provides a mechanism to ensure 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 naturally applies to mobility events, where the consumer may connect between multiple access points before a mobility event (make-before-break handoff). In such mobility event, the consumer is connected first to the previous access point, then to both the previous and next access points, then finally only to the next access points. With CCN, the consumer only sends interests on the available interfaces. By combining NC with CCN, requesting coded packets ensures that during the phase where it is connected to the previous and the next APs at the same time, it does not receive duplicate data, but does not miss on any content either. The consumer receives additional degrees of freedom with any innovative packet it receives on either interface. From this point of view, an effective interest forwarding strategy for obtaining innovative packets should be considered for consumer to achieve seamless mobility.

4.5. Security and Privacy

This subsection describes the requirement for security and privacy provided by NC in CCN/NDN, such as data integrity especially when intermediate nodes perform re-encoding, as in the case of hash restrictions for original data packets, and so on.

Network coding impacts the security mechanisms of CCN/NDN. In particular, CCN/NDN is designed to prevent modification of the Data packets. Because Data packets for a specific name can be self-authenticated, they can be validated on the delivery path, and can also be cached at untrusted intermediate nodes. Network coding may bring up issues if intermediate nodes are allowed to modify packets by performing additional network coding operations. In addition, if in-network caches store coded packets, they need to be able to validate that the packets are not compromised, so as to avoid cache...
pollution attacks. Without having all the packets in a generation, the cache cannot decode the packets to check if it is authenticated.

In CCN/NDN, content objects can be encrypted to support access control or privacy. If the coding information of coded packet are encrypted together with the payload (for instance, at source coding), the content requestor or forwarding nodes would incur extra computational overhead for decryption of the packet to interpret the coding information. With consideration for low computation overhead, some mechanism supporting both NC and access control/privacy should be considered.

5. Challenges

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

5.1. Adopting Convolutional Coding

Several block coding approaches have been proposed so far, but there is still no sufficient discussion and application of convolutional coding approach (e.g., sliding or elastic window coding) in CCN/NDN. Convolutional coding is often appropriate to situations where a fully or partially reliable delivery of continuous data flows is needed, especially when these data flows feature realtime constraints. As in [34] on an end-to-end basis, it would be advantageous for continuous content flow to adopt sliding window coding in CCN/NDN. In this case, the publisher needs to appropriately set coding parameters and let content requestor know the information, and content requestor needs to send interest (i.e., feedback information) about the data reception status. Since CCN/NDN advocates hop-by-hop communication, it would be worth discussing and investigating how convolutional coding can be applied in a hop-by-hop fashion and the benefits. In particular, assuming that NC could occur at intermediate nodes with some useful data packets stored in the CS as described in the previous section, both the encoding window and CS management would be required, and the feasibility and practicality should be considered.

5.2. Rate and Congestion Control

Adding redundancy using coded packets may cause further network congestion and adversely affect overall throughput performance. In particular, in a situation where 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 indispensable that each content flow cooperatively implements congestion control to adjust the consumed bandwidth to stabilize the network condition (i.e., to achieve low packet loss rate, delay, and
jitter). From this point of view, a router supported approach would be effective, but an effective deployment scenario is needed.

As described in Section 4.4, NC can contribute to seamless mobility by obtaining innovative packets without receiving duplicated packets through a virtual logical link to multiple copies of the content. To achieve seamless mobility while improving overall throughput or latency, an effective rate adaptation mechanism upon the virtual logical link is also challenging.

5.3. Security and Privacy

CCN/NDN introduces new security and privacy issues at the networking layer different from IP network, such as cache poisoning and pollution attack, DoS attack using interest packets, and so on.

NC could be utilized to mitigate some security or privacy issues CCN/NDN introduces. For instance, assuming that consumers can utilize multipath data retrieval and caching in CCN/NDN with NC, cache privacy and anonymity set for consumers can be improved as well as caching performance due to the diversity of caching content along different paths.

On the other hand, considering NC operations over CCN/NDN, the issues related to in-network caching add additional complexity. In order to avoid cache poisoning attack which tries to fill routers cache with polluted content, router needs to check whether or not the content is validated. However, in the case of performing NC and generating a new coded data at routers, a validation mechanism to accurately verify coded data as quickly as possible should be considered while maintaining in-network cache benefits (lower latency and network resource saving). If router can cache some valid coded data, it needs to put a great deal of thought into the effectiveness with respect to cache pollution attack, since coded data newly generated may be unpopular. Moreover, Denial of Service (DoS) attacks may target either the routers or the publishers performing NC to pose unnecessary coded data, impose higher NC computation load, and increase the number of PIT entries, which requires some careful considerations to avoid them.

NC also offers a new surface of attack; for instance, if the coding vector is exposed at the network layer, it would have to be protected (and validated) so as to avoid modifications by an attacker (and allow for verification) on the path of the packet.

In this context, from the perspective of both feasibility and practicability, a more effective approach with consideration for
security and privacy would be needed in order to accelerate the deployment of CCN/NDN with NC.

5.4. Routing Scalability

In CCN/NDN, a name-based routing protocol without a resolution process streamlines the routing process and reduces the overall latency. As in IP routing, the growth in the routing table size has become a concern. This may require a hierarchical naming scheme so as to improve the routing scalability by enabling aggregation of routing information. Moreover, it is a challenge that content requestors efficiently obtain linearly independent coded packets using multipath retrieval in a fully distributed manner, in order to fully leverage NC over CCN/NDN to improve throughput or reduce latency. This would require some efficient routing mechanism to appropriately set the FIB and also requires some efficient interest forwarding strategy. Such routing coordination may create routing scalability issues. From another NC perspective, as described Section 4.2.2, when issuing interests specifying unique names for each coded packet, consumers need in advance to know how to specify the names of the coded data through some specific name resolution scheme, and routers may need to appropriately set the FIBs. In this context, it would be challenging to achieve effective and scalable routing for interests requesting coded data as well as to simplify the routing process.

6. Security Considerations

This document does not impact the security of the Internet. Security considerations related to NC for CCN/NDN are described in the previous Section.

7. References

7.1. Normative References


7.2. Informative References


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@huawei.com
Generic Application Programming Interface (API) for Sliding Window FEC Codes

draft-roca-nwcrng-generic-fec-api-05

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 September 28, 2019.
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:

- 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;
- 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;
- 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;
- 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;
- 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);
- 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);
- 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 l repair symbols may be required to recover all the l 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.
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:

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

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

- session management (sender and receiver);
- encoding window management (sender and receiver);
- set/get/generate coding coefficients (sender and receiver);
- build coded symbol (sender only);
- 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;
typedef struct swif_encoder {
    swif_codepoint_t codepoint;
    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*, void*, esi_t);
    swif_status_t (*get_coding_window_information) (struct swif_encoder*, void*, esi_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;

typedef struct swif_decoder {
    swif_codepoint_t codepoint;
} swif_decoder_t;
/* 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 (*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.
swif_status_t   swif_encoder_set_parameters  (
    swif_encoder_t* enc,
    uint32_t        type,
    uint32_t        length,
    void*           value);

swif_status_t   swif_encoder_get_parameters  (
    swif_encoder_t* enc,
    uint32_t        type,
    uint32_t        length,
    void*           value);

enum {
    swif_ENCODER_GET_PARAM_ENCODER_STATISTICS = 1,
    swif_ENCODER_SET_PARAM_RLC_DENSITY_THRESHOLD
};
* @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! */

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

```c
/**
 * 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);
```

```c
/**
 * 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);
```

```c
/**
 * 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.

```c
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.

```c
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.

```c
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 (  

Decoder API proposal

/**
 * 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.

```c
/**
 * 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_esi   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_esi);
```

Roca (Ed.), et al. Expires September 28, 2019
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);

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:

- the application provides the list of coding coefficients to use  
  for the next build_repair_symbol();  
- 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();  
- 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;

```c
/**
 * 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 coefs_tab[].
* @return
/**
 * 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);

*/
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


8.2. Informative References


Authors’ Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Jonathan Detchart
ISAE - Supaero
France

EMail: jonathan.detchart@isae-supaero.fr