

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 3, 2018

N. Kuhn, Ed.
CNES
E. Lochin, Ed.
ISAE
H. Skinnemoen
AnsuR Technologies
S. Ghanem
Independent Senior Researcher
J. Bilbao
G. Peralta
Ikerlan
October 30, 2017

Network coding and satellites
draft-kuhn-nwcrg-network-coding-satellites-01

Abstract

This memo presents the current deployment of network coding in some satellite telecommunications systems along with a discussion on the multiple opportunities to introduce these technics at a wider scale.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Glossary	3
1.2. Requirements Language	3
2. A note on satellite topology	3
3. Status of network coding in actually deployed satellite systems	4
4. Opportunities for more network coding in satellite systems .	5
5. Deployability and related use cases	6
5.1. Network coding and VNF	6
5.2. Network coding and PEP	6
6. Acknowledgements	6
7. Contributors	6
8. IANA Considerations	7
9. Security Considerations	7
10. References	7
10.1. Normative References	7
10.2. Informative References	7
Authors' Addresses	7

1. Introduction

Network coding schemes are inherent part of the satellite systems, since the challenging physical layer require specific robustness to guarantee an efficient usage of the expensive radio resource. Further exploiting these schemes is an opportunity for a better end user experience along with a better exploitation of the scarce resource.

In this context, this memo aims at:

- o summing up the current deployment of network coding schemes;
- o identifying opportunities for further usage of network coding in satellite systems.

1.1. Glossary

The glossary of this memo is related to the network coding taxonomy document [I-D.irtf-nwcrg-network-coding-taxonomy].

The glossary is extended as follows:

- o XX: XX

1.2. 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].

2. A note on satellite topology

The objective of this section is to provide a generic description of the components composing a generic satellite system and their interaction. It provides a high level description of a multi-gateway satellite network. Figure 1 shows an example of a multigateway satellite system. It is worth pointing out that some functional blocks aggregate the traffic coming from multiple users, and thus are an opportunity for including network coding.

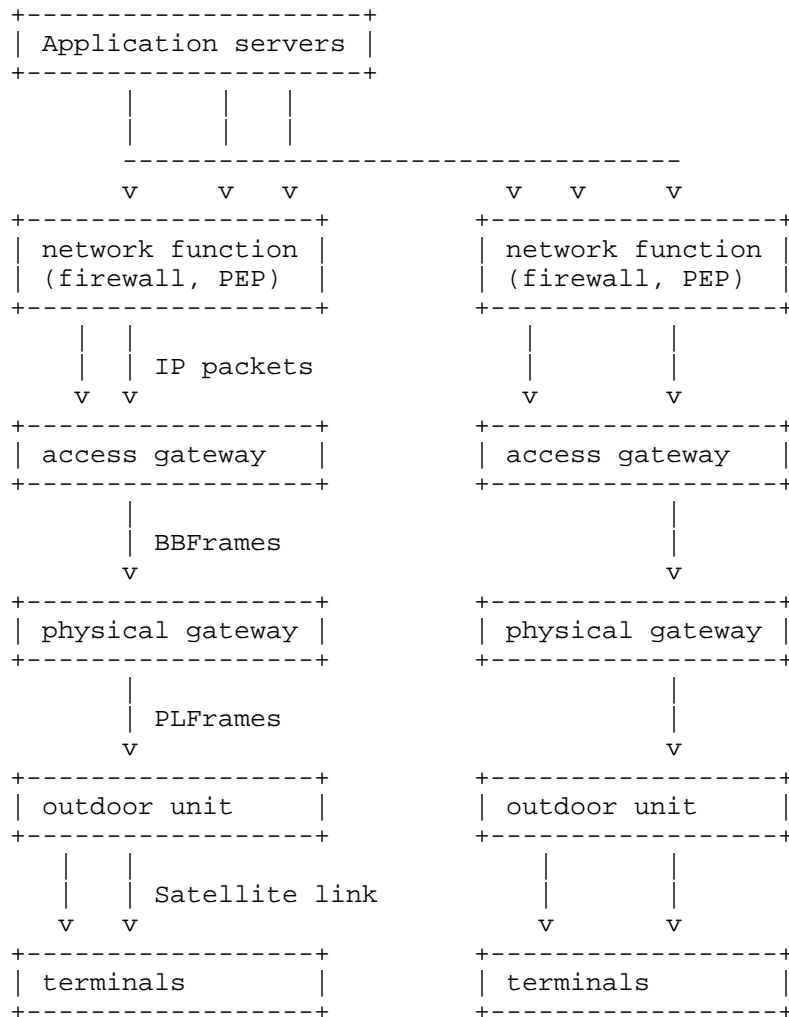


Figure 1: Data plane functions in a generic satellite multi-gateway system

3. Status of network coding in actually deployed satellite systems

Figure 2 presents the status of the network coding deployment in satellite systems. The information is based on the taxonomy document [I-D.irtf-nwcrp-network-coding-taxonomy] and the notations are the following: End-to-End Coding (E2E), Network Coding (NC), Intra-Flow Coding (IntraF), Inter-Flow Coding (InterF), Single-Path Coding (SP) and Multi-Path Coding (MP).

X1 embodies the source coding that could be used at application level for video streaming on a broadband access. X2 embodies the physical layer that is applied on the PLFRAME to have an optimal usage of the satellite capacity.

	Upper Appl.	Middle ware	Communication layers	
	Source coding	Network AL-FEC	Packetization UDP/IP	PHY layer
E2E	X1			
NC				
IntraF	X1			
InterF				X2
SP	X1			X2
MP				

Figure 2: Network coding and satellite systems

4. Opportunities for more network coding in satellite systems

This section extends Section 3 by presenting the opportunities for more network coding in satellite systems.

These opportunities are further detailed in Section 5 and listed in this section:

- o (1) two way relay channel mode;
- o (2) reliable multicast;
- o (3) improving random access;
- o (4) network coding and hybrid access;

We propose to include some of the identified opportunities in the Figure 3.

	Upper Appl.	Middle ware	Communication layers		
	Source coding	Network AL-FEC	Packetization UDP/IP	PHY layer	
E2E	X1		(4)		
NC		(1)	(1)(2)(3)(4)		
IntraF	X1		(2)(4)		
InterF		(1)	(1)(3)		X2
SP	X1	(1)	(1)(3)		X2
MP			(2)		

Figure 3: Opportunities for more network coding and satellite systems

Opportunities for more network coding in SATCOM seems to be more relevant at the middle ware or at the communication layer levels.

5. Deployability and related use cases

This section details use-cases where the usage of network coding schemes could improve the overall system and the deployability of the opportunities that are provided in Section 4.

5.1. Network coding and VNF

Related to the foreseen virtualized network infrastructure, the network coding schemes could be proposed as VNF and their deployability enhanced.

5.2. Network coding and PEP

Related to the impact and integration of network coding in Proxy-Enhanced-Proxy RFC 3135 [RFC3135] architecture. In particular how network coding can be integrated inside a PEP with QoS scheduler as defined, for instance, in RFC 5865 [RFC5865].

6. Acknowledgements

7. Contributors

Many thanks to

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

This document, by itself, presents no new privacy nor security issues.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, DOI 10.17487/RFC3135, June 2001, <<https://www.rfc-editor.org/info/rfc3135>>.
- [RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <<https://www.rfc-editor.org/info/rfc5865>>.

10.2. Informative References

- [I-D.irtf-nwcrp-network-coding-taxonomy] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., samah.ghanem@gmail.com, s., Lochin, E., Masucci, A., Montpetit, M., Pedersen, M., Peralta, G., Roca, V., Saxena, P., and S. Sivakumar, "Network Coding Taxonomy", draft-irtf-nwcrp-network-coding-taxonomy-05 (work in progress), July 2017.

Authors' Addresses

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

Phone: 0033561273213
Email: nicolas.kuhn@cnes.fr

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

Email: emmanuel.lochin@isae.fr

Harald Skinnemoen
AnsuR Technologies
Martin Linges Vei 25
Fornebu 1364
Norway

Email: harald@ansur.no

Samah A. M. Ghanem
Independent Senior Researcher
West Bank
Palestine

Email: samah.ghanem@gmail.com

Josu Bilbao
Ikerlan
Spain

Email: JBilbao@ikerlan.es

Goiuri Peralta
Ikerlan
Spain

Email: gperalta@ikerlan.es

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

N. Kuhn, Ed.
CNES
E. Lochin, Ed.
ISAE-SUPAERO
July 2, 2018

Network coding and satellites
draft-kuhn-nwcrg-network-coding-satellites-05

Abstract

This memo presents the current deployment of network coding in some satellite telecommunications systems along with a discussion on the multiple opportunities to introduce these techniques at a wider scale.

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 January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Glossary	3
1.2. Requirements Language	3
2. A note on satellite topology	4
3. Status of network coding in actually deployed satellite systems	6
4. Details on the use cases	6
4.1. Two way relay channel mode	7
4.2. Reliable multicast	7
4.3. Hybrid access	8
4.4. Dealing with varying capacity	9
4.5. Improving the gateway handovers	10
4.6. Delay/Disruption Tolerant Networks	10
5. Discussion on the deployability	11
6. Conclusion	12
7. Acknowledgements	12
8. Contributors	12
9. IANA Considerations	12
10. Security Considerations	12
11. References	13
11.1. Normative References	13
11.2. Informative References	13
Authors' Addresses	16

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 had to add to cope from link impairments, without reducing the good-put when the channel quality is high. Generally speaking, enough redundancy is added so as to guarantee a Quasi-Error Free transmission; however, there are cases where the physical layer could hardly recover the transmission losses (e.g. with a mobile user) and layer 2 (or above) re-transmissions induce an at least 500 ms delay with a geostationary satellite. Further exploiting network coding schemes at higher OSI-layers is an opportunity for releasing constraints on the physical layer and improve the performance of SATCOM systems when the physical layer is challenged. We have noticed an active research activity on how network coding and SATCOM in the past. That being said, not much has actually made it to industrial developments. In this context, this memo aims at:

- o summing up the current deployment of network coding schemes over LEO and GEO satellite systems;

- o identifying opportunities for further usage of network coding in these systems.

1.1. Glossary

The glossary of this memo is related to the network coding taxonomy document [I-D.irtf-nwcrg-network-coding-taxonomy].

The glossary is extended as follows:

- o BBFRAME: Base-Band FRAME - satellite communication layer 2 encapsulation work as follows: (1) each layer 3 packet is encapsulated with a Generic Stream Encapsulation (GSE) mechanism, (2) GSE packets are gathered to create BBFRAMEs, (3) BBFRAMEs contain information related to how they have to be modulated (4) BBFRAMEs are forwarded to the physical layer;
- o CPE: Customer Premise Equipment;
- o DTN: Delay/Disruption Tolerant Network;
- o EPC: Evolved Packet Core;
- o ETSI: European Telecommunications Standards Institute;
- o PEP: Performance Enhanced Proxy - a typical PEP for satellite communications include compression, caching and TCP acceleration;
- o PLFRAME: Physical Layer FRAME - modulated version of a BBFRAME with additional information (e.g. related to synchronization);
- o SATCOM: generic term related to all kind of SATellite COMMunications systems;
- o UMTRAN: Universal Mobile Terrestrial Radio Access Network;
- o QoS: Quality-of-Service;
- o QoE: Quality-of-Experience;
- o VNF: Virtualized Network Function.

1.2. 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].

2. A note on satellite topology

The objective of this section is to provide both a generic description of the components composing a generic satellite system and their interaction. It provides a high level description of a multi-gateway satellites network. There exist 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. This memo lays on SATCOM systems dedicated to Internet access that follows the DVB-S2/RCS2 standards.

In this context, Figure 1 shows an example of a multi-gateway satellite system. More details on a generic SATCOM ground segment architecture for a bi-directional Internet access can be found in [SAT2017]. We propose a multi-gateway system since some of the use-cases described in this document require multiple gateways. 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 trade-off discussions can be found in [SAT2017].

It is worth noting that some functional blocks aggregate the traffic coming from multiple users. Even if network coding schemes could be applied to any individual traffic, it could also work on a aggregate.

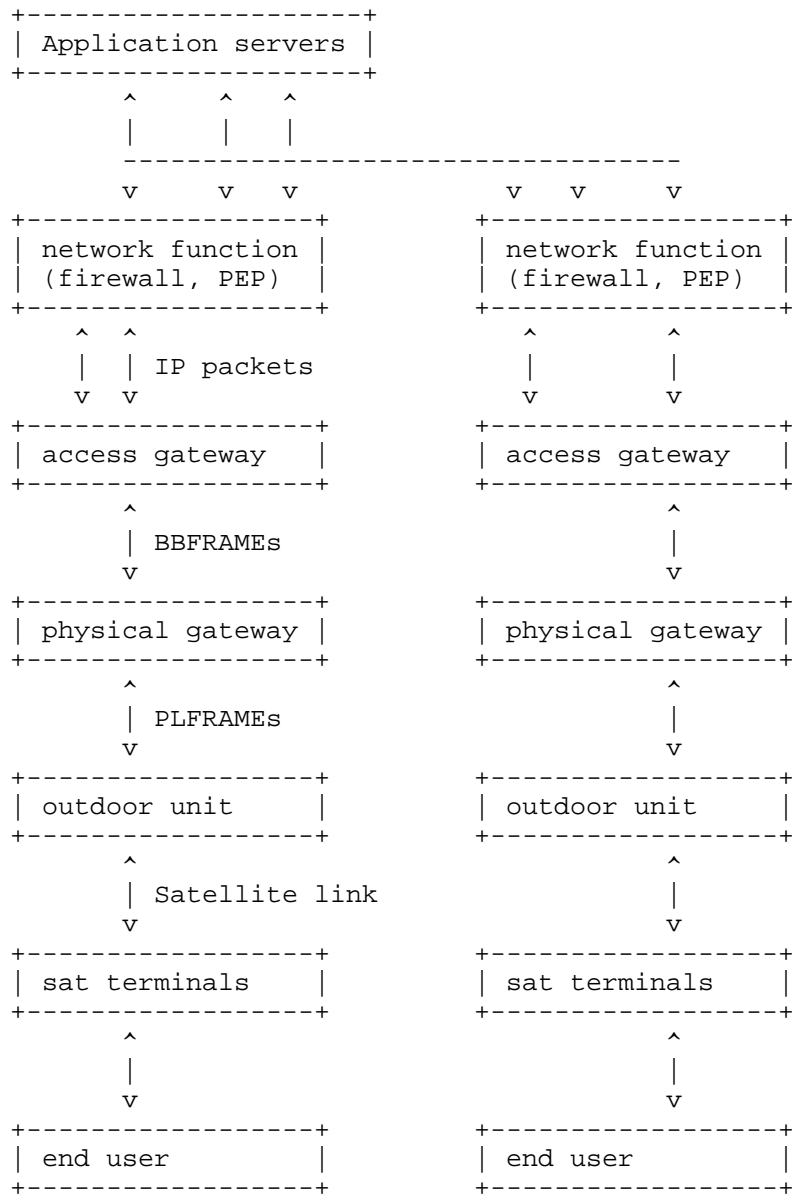


Figure 1: Data plane functions in a generic satellite multi-gateway system

3. Status of network coding in actually deployed satellite systems

Figure 2 presents the status of the network coding deployment in satellite systems. The information is based on the taxonomy document [I-D.irtf-nwcr-g-network-coding-taxonomy] and the notations are the following: End-to-End Coding (E2E), Network Coding (NC), Intra-Flow Coding (IntraF), Inter-Flow Coding (InterF), Single-Path Coding (SP) and Multi-Path Coding (MP).

X1 embodies the source coding that could be used at application level for instance: for video streaming on a broadband access. X2 embodies the physical layer, applied to the PLFRAME, to optimize the satellite capacity usage. Furthermore, at the physical layer and when random accesses are exploited, FEC mechanisms are exploited.

	Upper Appl.	Middle ware	Communication layers	
	Source coding	Network AL-FEC	Packetization UDP/IP	PHY layer
E2E	X1			
NC				
IntraF	X1			
InterF				X2
SP	X1			X2
MP				

Figure 2: Network coding in current satellite systems

4. Details on the use cases

This section details use-cases where network coding schemes could improve the overall performance of a SATCOM system (e.g. considering a more efficient usage of the satellite resource, delivery delay, delivery ratio).

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

4.1. Two way relay channel mode

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

Satellite terminal A (resp. B) transmits a flow A (resp. B) to a server hosting NC capabilities, which forward a combination of the two flows to both terminals. This results in non-negligible bandwidth saving and has been demonstrated at ASMS 2010 in Cagliari [ASMS2010]. Moreover, with On-Board Processing satellite payloads, the network coding operations could be done at the satellite level, thus reducing the end-to-end delay of the communication.

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

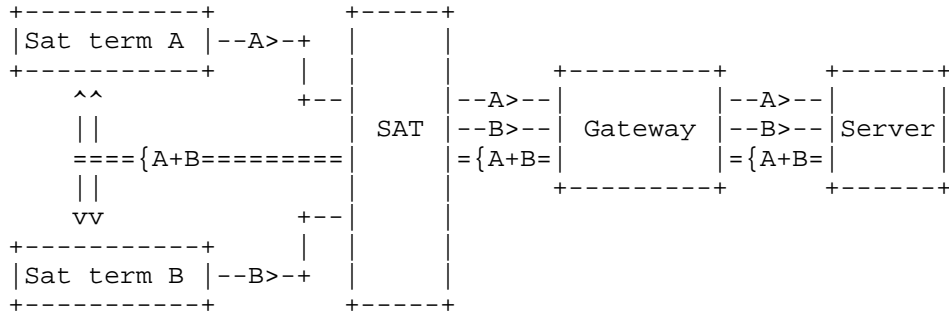


Figure 3: Network architecture for two way relay channel with NC

4.2. Reliable multicast

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.

A multicast flow (M) is forward to both satellite terminals A and B. On the uplink, terminal A (resp. B) does not acknowledge the packet N_i by sending a L_i signal (resp. N_j , L_j) and either the access gateway or the multicast server includes the missing packets in the multicast flow so that the information transfer is reliable. This could be achieved by using NACK-Oriented Reliable Multicast (NORM) [RFC5740]. However, NORM does not consider other network coding schemes such as sliding window encoding described in [I-D.irtf-nwcrg-network-coding-taxonomy].

-Li>- : packet indicated the loss of packet i of a multicast flow
 = {M== : multicast flow including the missing packets

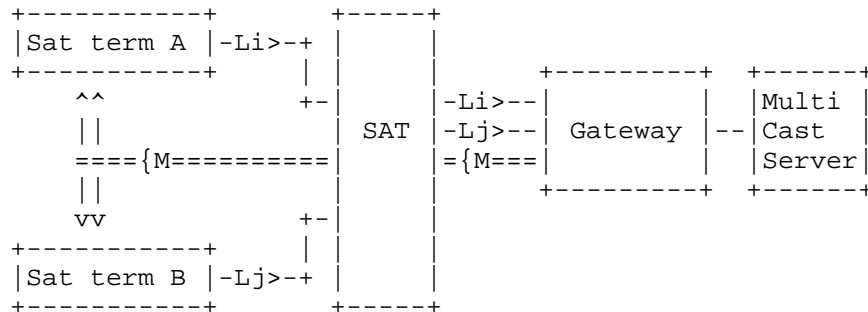


Figure 4: Network architecture for a reliable multicast with NC

4.3. Hybrid access

This use-case considers the use of multiple path management with network coding at the transport level to increase the reliability and/or the total bandwidth (using multiple path does not guarantee an improvement of both the reliability and the total bandwidth). 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]. It is worth nothing that this kind of architecture is also discussed in the MPTCP working group [I-D.boucadair-mptcp-dhc].

To cope from packet loss (due to either end-user movements or physical layer impairments), network coding could be introduced in both the CPE and at the concentrator.

->- : bidirectional link

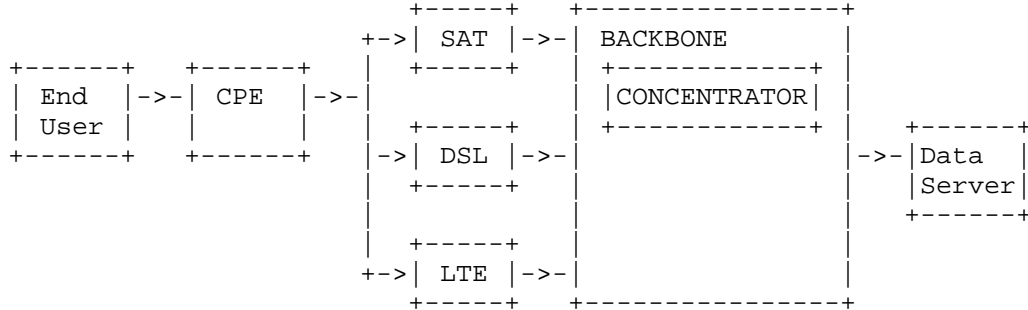


Figure 5: Network architecture for an hybrid access using NC

4.4. Dealing with varying capacity

This use-case considers the usage of network coding to overcome cases where the wireless link characteristics quickly change overtime and where the physical layer codes could not be made robust in time. This is particularly relevant when end users are moving and the channel shows important variations [IEEEVT2001].

The architecture is recalled in Figure 6. The network coding schemes could be applied at the access gateway or the network function levels to increase the reliability of the transmission. This use-case is mostly relevant for when mobile users are considered or when the chosen band induce 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 network coding is different. Then, depending on the OSI level at which network coding is applied, the impact on the satellite terminal is different: network coding may be applied on IP packets or on layer-2 proprietary format packets.

->- : bidirectional link

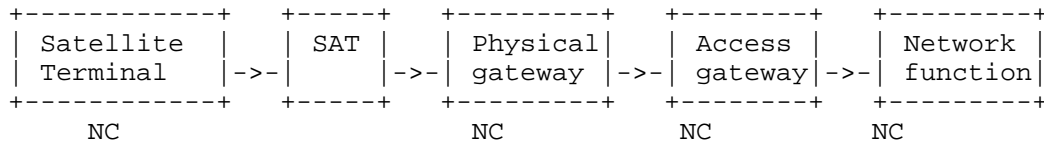


Figure 6: Network architecture for dealing with varying link characteristics with NC

4.5. 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, this may result in packet loss. During these critical phases, network coding can be added to improve the reliability of the transmission and propose a seamless gateway handover. In this case, the network coding could be applied at either the access gateway or the network function block. The entity responsible for taking the decision to change the communication gateway and changing the routes is the control plane manager; this entity exploits a management interface.

An example architecture for this use-case is showed in Figure 7. It is worth noting that depending on the ground architecture [I-D.chin-nfvrg-cloud-5g-core-structure-yang] [SAT2017], some equipment might be communalised.

->- : bidirectional link
! : management interface

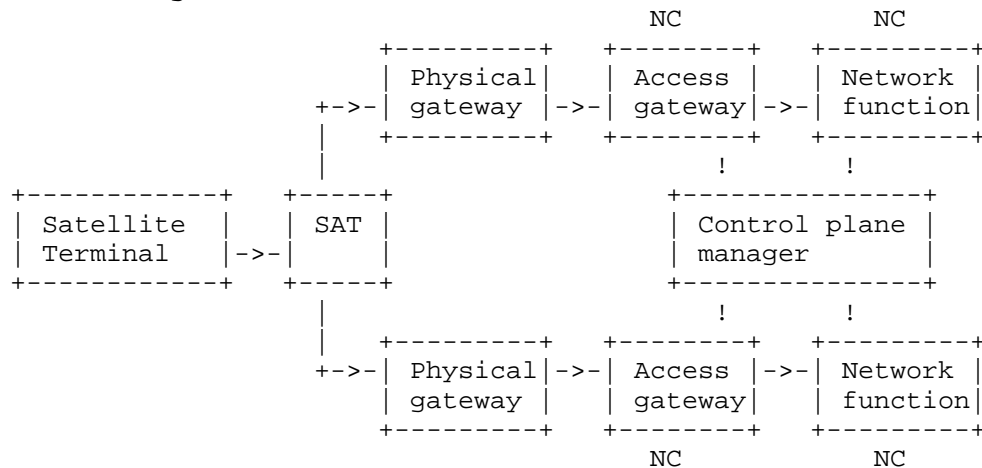


Figure 7: Network architecture for dealing with gateway handover schemes with NC

4.6. Delay/Disruption Tolerant Networks

Establishing communications from terrestrial gateways to aerospace components is a challenge due to the distances involved. As a matter of fact, reliable end-to-end (E2E) communications over such links must cope with long delay and frequent link disruptions. 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. DTN can also be seen as an alternative solution to cope with satellite communications usually managed by PEP. Therefore, the transport of data over such networks requires the use of replication, erasure codes and multipath protocol schemes [WANG05] [ZHANG06] to improve the bundle delivery ratio and/or delivery delay. For instance, transport protocols such as LTP [RFC5326] for long delay links with connectivity disruptions, use Automatic Repeat-reQuest (ARQ) and unequal error protection to reduce the amount of non-mandatory re-transmissions. The work in [TOURNOUX10] proposed upon LTP a robust streaming method based on an on-the-fly coding scheme, where encoding and decoding procedures are done at the source and destination nodes, respectively. However, each link path loss rate may have various order of magnitude and re-encoding at an intermediate node to adapt the redundancy can be mandatory to prevent transmission wasting. This idea has been put forward in [I-D.zinky-dtnrg-random-binary-fec-scheme] and [I-D.zinky-dtnrg-erasure-coding-extension], where the authors proposed an encoding process at intermediate DTN nodes to explore the possibilities of Forward Error Correction (FEC) schemes inside the bundle protocol [RFC5050]. Another proposal is the use of erasure coding inside the CCSDS (Consultative Committee for Space Data Systems) architecture [COLA11]. The objective is to extend the CCSDS File Delivery Protocol (CFDP) [CCSDS-FDP] with erasure coding capabilities where a Low Density Parity Check (LDPC) [RFC6816] code with a large block size is chosen. Recently, on-the-fly erasure coding schemes [LACAN08] [SUNDARARAJAN08] [TOURNOUX11] have shown their benefits in terms of recovery capability and configuration complexity compared to traditional FEC schemes. Using a feedback path when available, on-the-fly schemes can be used to enable E2E reliable communication over DTN with adaptive re-encoding as proposed in [THAI15].

5. Discussion on the deployability

This section discusses the deployability of the use-cases detailed in Section 4.

SATCOM systems typically feature Performance Enhancement Proxy RFC 3135 [RFC3135] which could be relevant to host network coding mechanisms and thus support the use-cases that have been discussed in Section 4. In particular the discussion on how network coding can be integrated inside a PEP with QoS scheduler has been proposed in RFC 5865 [RFC5865].

Related to the foreseen virtualized network infrastructure, the network coding schemes could be proposed as VNF and their deployability enhanced. The architecture for the next generation of SATCOM ground segments would rely on a virtualized environment. This trend can also be seen, making the discussions on the deployability of network coding in SATCOM extendable to other deployment scenarios [I-D.chin-nfvrg-cloud-5g-core-structure-yang]. As one example, the network coding VNF functions deployment in a virtualized environment is presented in [I-D.vazquez-nfvrg-netcod-function-virtualization].

6. Conclusion

This document presents the current deployment of network coding in some satellite telecommunications systems along with a discussion on the multiple opportunities to introduce these techniques at a wider scale.

Even if this document focuses on satellite systems, it is worth pointing out that the some scenarios proposed may be relevant to other wireless telecommunication systems. As one example, the generic architecture proposed in Figure 1 may be mapped to cellular networks as follows: the 'network function' block gather 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.

7. Acknowledgements

Many thanks to Tomaso de Cola, Vincent Roca and Marie-Jose Montpetit.

8. Contributors

Tomaso de Cola, Marie-Jose Montpetit.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

This document, by itself, presents no new privacy nor security issues.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

11.2. Informative References

- [ASMS2010] De Cola, T. and et. al., "Demonstration at opening session of ASMS 2010", ASMS , 2010.
- [CCSDS-FDP] "CCSDS File Delivery Protocol, Recommendation for Space Data System Standards", CCSDS 727.0-B-4, Blue Book num. 3, 2007.
- [COLA11] De Cola, T., Paolini, E., Liva, G., and G. Calzolari, "Reliability options for data communications in the future deep-space missions", Proceedings of the IEEE vol. 99 issue 11, 2011.
- [ETSITR2017] "Satellite Earth Stations and Systems (SES); Multi-link routing scheme in hybrid access network with heterogeneous links", ETSI TR 103 351, 2017.
- [I-D.boucadair-mptcp-dhc] Boucadair, M., Jacquenet, C., and T. Reddy, "DHCP Options for Network-Assisted Multipath TCP (MPTCP)", draft-boucadair-mptcp-dhc-08 (work in progress), October 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.irtf-nwcr-g-network-coding-taxonomy] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., samah.ghanem@gmail.com, s., Lochin, E., Masucci, A., Montpetit, M., Pedersen, M., Peralta, G., Roca, V., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", draft-irtf-nwcr-g-network-coding-taxonomy-08 (work in progress), March 2018.

- [I-D.vazquez-nfvrg-netcod-function-virtualization]
Vazquez-Castro, M., Do-Duy, T., Romano, S., and A. Tulino,
"Network Coding Function Virtualization", draft-vazquez-
nfvrg-netcod-function-virtualization-02 (work in
progress), November 2017.
- [I-D.zinky-dtnrg-erasure-coding-extension]
Zinky, J., Caro, A., and G. Stein, "Bundle Protocol
Erasure Coding Extension", draft-zinky-dtnrg-erasure-
coding-extension-00 (work in progress), August 2012.
- [I-D.zinky-dtnrg-random-binary-fec-scheme]
Zinky, J., Caro, A., and G. Stein, "Random Binary FEC
Scheme for Bundle Protocol", draft-zinky-dtnrg-random-
binary-fec-scheme-00 (work in progress), August 2012.
- [IEEEVT2001]
Fontan, F., Vazquez-Castro, M., Cabado, C., Garcia, J.,
and E. Kubista, "Statistical modeling of the LMS channel",
BEER Transactions on Vehicular Technology vol. 50 issue 6,
2001.
- [LACAN08] Lacan, J. and E. Lochin, "Rethinking reliability for long-
delay networks", International Workshop on Satellite and
Space Communications , October 2008.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z.
Shelby, "Performance Enhancing Proxies Intended to
Mitigate Link-Related Degradations", RFC 3135,
DOI 10.17487/RFC3135, June 2001,
<<https://www.rfc-editor.org/info/rfc3135>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst,
R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant
Networking Architecture", RFC 4838, DOI 10.17487/RFC4838,
April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol
Specification", RFC 5050, DOI 10.17487/RFC5050, November
2007, <<https://www.rfc-editor.org/info/rfc5050>>.
- [RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider
Transmission Protocol - Specification", RFC 5326,
DOI 10.17487/RFC5326, September 2008,
<<https://www.rfc-editor.org/info/rfc5326>>.

- [RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, DOI 10.17487/RFC5740, November 2009, <<https://www.rfc-editor.org/info/rfc5740>>.
- [RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <<https://www.rfc-editor.org/info/rfc5865>>.
- [RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <<https://www.rfc-editor.org/info/rfc6816>>.
- [SAT2017] Ahmed, T., Dubois, E., Dupe, JB., Ferrus, R., Gelard, P., and N. Kuhn, "Software-defined satellite cloud RAN", Int. J. Satell. Commun. Network. vol. 36, 2017.
- [SUNDARARAJAN08] Sundararajan, J., Shah, D., and M. Medard, "ARQ for network coding", IEEE Int. Symp. on Information Theory, July 2008.
- [THAI15] Thai, T., Chaganti, V., Lochin, E., Lacan, J., Dubois, E., and P. Gelard, "Enabling E2E reliable communications with adaptive re-encoding over delay tolerant networks", Proceedings of the IEEE International Conference on Communications, June 2015.
- [TOURNOUX10] Tournoux, P., Lochin, E., Leguay, J., and J. Lacan, "On the benefits of random linear coding for unicast applications in disruption tolerant networks", Proceedings of the IEEE International Conference on Communications, 2010.
- [TOURNOUX11] Tournoux, P., Lochin, E., Lacan, J., Bouabdallah, A., and V. Roca, "On-the-fly erasure coding for real-time video applications", IEEE Trans. on Multimedia vol. 13 issue 4, August 2011.
- [WANG05] Wang, Y. and et. al., "Erasure-coding based routing for opportunistic networks", Proceedings of the ACM SIGCOMM workshop on Delay-tolerant networking, 2005.

[ZHANG06] Zhang, X. and et. al., "On the benefits of random linear coding for unicast applications in disruption tolerant networks", Proceedings of the 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks , 2006.

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-supero.fr

Network Coding Research Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

K. Matsuzono
H. Asaeda
NICT
C. Westphal
Huawei
March 5, 2018

Network Coding for Content-Centric Networking / Named Data Networking:
Requirements and Challenges
draft-matsuzono-nwcr-g-nwc-ccn-reqs-01

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
2.1. Definitions	3
2.2. NDN/CCN Background	5
3. Advantage given by NC and CCN/NDN	6
4. Requirements	7
4.1. Content Naming	7
4.2. Transport	8
4.2.1. Scope of Network Coding	9
4.2.2. Consumer Operation	9
4.2.3. Router Operation	10
4.2.4. Publisher Operation	11
4.3. In-network Caching	11
4.4. Seamless Mobility	12
4.5. Security and Privacy	12
5. Challenges	13
5.1. Adopting Convolutional Coding	13
5.2. Rate and Congestion Control	13
5.3. Security and Privacy	14
5.4. Routing Scalability	14
6. Security Considerations	14
7. References	14
7.1. Normative References	14
7.2. Informative References	14
Authors' Addresses	17

1. Introduction

Information-Centric Networks in general, and Content-Centric Networking (CCN) [15] or Named Data Networking (NDN) [16] 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 heterogeneous 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].

NC aggregates multiple packets with parts of the same content together, 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 evolved 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 (described in more detail later on). NC has already been implemented for information/content dissemination (e.g. [6][7][8]). NC provides CCN/NDN with the highly beneficial potential to effectively disseminate information in a completely independent and decentralized manner. [9] first suggested to exploit NC techniques to enhance key system performances in ICN, and others have considered NC in ICN use cases such as content dissemination [10], seamless mobility [11], joint caching and network coding [12][13], low-latency video streaming [14], etc.

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 recent activities in the Network Coding Research Group [18].

- o Random Linear Coding (RLC): Particular case of Linear Coding using a set of random coding coefficients.
- o Generation, or (IETF) Block: With Block Codes, the set of content data that are logically grouped into a Block, before doing encoding.
- o Generation Size: With Block Codes, the number k of content data belonging to a Block.
- o Encoding Vector: A set of coding coefficients used to generate a certain coded packet through linear coding. The number of nonzero coefficients in the Coding Vector defines its density
- o 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.
- o Finite Field size: The number of elements in a finite field. For example the binary extension field $\{0..2^m-1\}$ has size $q=2^m$.
- o 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.
- o 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.
- o 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").
- o 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.

- o Consumer: A node requesting content. It initiates communication by sending an interest packets.
- o Publisher: A node providing content. It originally creates or owns the content.
- o Forwarding Information Base (FIB): A lookup table in a content router containing the name prefix and corresponding destination interface to forward the interest packets.
- o 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.
- o Content Store (CS): A storage space for a router to cache content objects. It is also known as in-network cache.
- o Content Object: A unit of content data delivered through the CCN/NDN network.
- o 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 request 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 later versions of CCN, the interest must carry a full name, while in NDN 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 send back to the requester(s) using the trail of PIT entries; intermediate node 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. Advantage given 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 [17], 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, cannot provide reliable and robust content dissemination. 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. Furthermore, NC may further enhance CCN/NDN security [23]. In this context, it should be natural that there is much room for considering NC integration into CCN/NDN transport exploiting in-network caching and multi-path transmission [9] and seamless mobility [11] [28].

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 [30]. 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 [24].

However, it requires fully adjustable and specific name-based routing mechanism for CCN/NDN, and an intense computational task for central coordination. In the case of non-coherent NC that often utilizes RLC, they do not need to know network topology and intermediate coding operations. Since non-coherent NC works in a completely independent and decentralized manner, this approach is more feasible especially in the large scale use cases that are intended with CCN/NDN. This document thus focuses on non-coherent NC with RLC.

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 [19]. 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 degrades 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/FIB/CS operations need a unique name to identify the coded data. As a way of naming coded packet, the encoding 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 encoding 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 FIB/PIT/CS as for original source packets. However, such a naming way requires the consumer 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. From this point of view, it could shift the generation of the encoding vector from the content producer onto the content requester.

If a naming schema such as above is used, it would be valuable to reconsider whether Interest 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 requestor 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 requestor receives (also called decoding matrix) as in [9]). However, it may incur a largely increased size of TLV header. Without such coding information, the forwarding node would need to maintain some records regarding interest packets sent before, in order to provide new degrees of freedom.

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). This however would preclude network coding on packets without prior decoding them (for instance, in the CS of forwarding nodes). 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, as described above, some mechanism needs for the content requestor to obtain innovative coded packets. It would also require some mechanism to insert the multiple innovative packets into the CS. 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.

4.2. Transport

The pull-based request-response feature of CCN/NDN is the fundamental principle of its transport layer; one Interest retrieves at most one Data packet. 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.

4.2.1. Scope of Network Coding

It should be discussed whether the network can update 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 attain NC benefits associated with in-network caching, consumers need 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. Issuing such an interest is possible by using optional TLV (Type Length Value) header contained in Interest TLV packet format which allows network elements to add or modify information on the fly. Consumer can put an instruction into it, and for instance, if routers detect that it is better for consumer to get coded packets rather than original packets, routers can modify it to do so. After receiving interests having the instruction in optional header, the router with useful coded packets forward them.

As another solution, consumer issues interests specifying unique names for each coded packets. In this case, a unified naming scheme considering both original and coded packets is required. Moreover, in the case of NC end-to-end approach, publishers need to get feedback from the corresponding receivers to adjust some coding parameters. To deal with this, a receiver may have to request a specific interest name to reach the corresponding publisher and put required information into the optional header.

4.2.3. Router Operation

Routers need to appropriately handle PIT entries to accommodate interests for coded packets as well as original packets. Moreover, in order to decode as necessary, nodes need to know the coding vector used for each coded packet (note: since all the data for a specific content may not come through the same path/network, intermediate nodes may never be able to decode). In a typical case, the coding vector used for each coded packet is attached to the header of coded data. In regard to this point, the generation size (also called block size) for NC should be set to a reasonable value so that the total coded packet size including header needed for expressing the coding vector information and data message fits into the allowable packet size. It may be useful to use compression techniques for coding vectors [20][21].

Router may try to forward useful independent coded packets toward downstream nodes in order to respond to received interests for coded packets. 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 packets has a strong probability of making independent coded packets [24]. If router does not have enough cached packets to newly produce independent coded packets, it relays received interests to upstream nodes to receive a new original or independent coded packet and pass it to downstream nodes. 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 there is a tradeoff between NC encoding/decoding calculation cost and cache capacity, and the usage efficacy of re-encoding or decoding at router, router should need to determine how to response 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.

Some proposed schemes [10] require that the router maintain a tally of the interests for a specific name and generation, 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.

To enable fast loss recovery cooperating with in-network caching, a transport mechanism of in-network loss detection and recovery [28][14] at router as well as consumer-driven mechanism should be considered.

4.2.4. Publisher Operation

The procedure for splitting an overall content into small content objects is responsible for 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 encoding 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 encoding vectors after receiving interests specifying them. In the former case, although content requestors cannot flexibly specify an encoding 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. According to application requirement for latency, such NC operation strategy should be considered.

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 [29], 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 [22] [25] [26]. 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. 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. 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] [28] and clarify the requirements. A key feature of CCN/NDN is that it is sessionless and that multiple interests can be send to different copies of the content in parallel. CCN/NDN enables a consumer to retrieve the content from multiple sources that are distributed and asynchronous.

In this context, network coding provide a mechanism to ensure that the Interests sent to multiple copies of the content retrieve innovative packets, even in the case of packet losses on some of the paths/networks to these copies. NC adds a reliability layer to CCN in a distributed and asynchronous manner. One key benefit is that the link between the consumer and the multiple copies acts as a virtual logical link, upon which rate adaptation mechanism can be performed.

This naturally applies to mobility event, 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. Requesting network 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. By combining NC with CCN, the consumer receives additional degrees of freedom with any innovative packet it receives on either interface.

Further discussion is [TBD].

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. Intermediate nodes may also be caching network coded packets without having the ability to perform validation of the content and therefore open themselves to cache pollution attacks.

In CCN/NDN, content objects can be encrypted to support access control or privacy. If the coding information of coded packet is included in the encrypted data payload, extra computational overhead occurs.

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 [31] 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).

5.3. Security and Privacy

A variety of security and privacy concerns would exist in NC and CCN/NDN. This subsection focuses on the description of security and privacy challenges related to NC for CCN/NDN. [TBD]

5.4. Routing Scalability

This subsection focuses on the challenges of routing mechanisms such as scalability and protocol overhead, and so on.

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

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

7.2. Informative References

- [2] Cai, N. and R. Yeung, "Secure network coding", Proc. International Symposium on Information Theory (ISIT), IEEE, June 2002.
- [3] Lima, L., Gheorghiu, S., Barros, J., Mdard, M., and A. Toledo, "Secure Network Coding for Multi-Resolution Wireless Video Streaming", IEEE Journal of Selected Area (JSAC), vol. 28, no. 3, April 2002.
- [4] Gkantsidis, C. and P. Rodriguez, "Cooperative Security for Network Coding File Distribution", Proc. Infocom, IEEE, April 2006.
- [5] Vilea, J., Lima, L., and J. Barros, "Lightweight security for network coding", Proc. ICC, IEEE, May 2008.

- [6] Dimarkis, A., Godfrey, P., Wu, Y., Wainwright, M., and K. Ramchandran, "Network Coding for Distributed Storage Systems", IEEE Trans. Information Theory, vol. 56, no.9, September 2010.
- [7] Gkantsidis, C. and P. Rodriguez, "Network coding for large scale content distribution", Proc. Infocom, IEEE, March 2005.
- [8] Seferoglu, H. and A. Markopoulou, "Opportunistic Network Coding for Video Streaming over Wireless", Proc. Packet Video Workshop (PV), IEEE, November 2007.
- [9] Montpetit, M., Westphal, C., and D. Trossen, "Network Coding Meets Information-Centric Networking: An Architectural Case for Information Dispersion Through Native Network Coding", Proc. Workshop on Emerging Name-Oriented Mobile Networking Design (NoM), ACM, June 2012.
- [10] Saltarin, J., Bourtsoulatz, E., Thomos, N., and T. Braun, "NetCodCCN: a network coding approach for content-centric networks", Proc. Infocom, IEEE, April 2016.
- [11] Ramakrishnan, A., Westphal, C., and J. Saltarin, "Adaptive Video Streaming over CCN with Network Coding for Seamless Mobility", Proc. International Symposium on Multimedia (ISM), IEEE, December 2016.
- [12] Wang, J., Ren, J., Lu, K., Wang, J., Liu, S., and C. Westphal, "An Optimal Cache Management Framework for Information-Centric Networks with Network Coding", Proc. Networking Conference, IFIP/IEEE, June 2014.
- [13] Wang, J., Ren, J., Lu, K., Wang, J., Liu, S., and C. Westphal, "A Minimum Cost Cache Management Framework for Information-Centric Networks with Network Coding", Computer Networks, Elsevier, August 2016.
- [14] Matsuzono, K., Asaeda, H., and T. Turletti, "Low Latency Low Loss Streaming using In-Network Coding and Caching", Proc. Infocom, IEEE, May 2017.
- [15] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", Proc. CoNEXT, ACM, December 2009.

- [16] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Claffy, K., Crowley, P., Papadopoulos, C., Wang, L., and B. Zhang, "Named data networking", ACM Comput. Commun. Rev., vol. 44, no. 3, July 2014.
- [17] Koetter, R. and M. Medard, "An Algebraic Approach to Network Coding", IEEE/ACM Trans. on Networking, vol. 11, no 5, Oct. 2003.
- [18] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Lochin, E., Masucci, A., Montpetit, M., Pedersen, M., Peralta, G., Roca, V., Saxena, P., and S. Sivakumar, "Network Coding Taxonomy", draft-irtf-nwcrg-network-coding-taxonomy-05 (work in progress), September 2017.
- [19] Kutscher, et al., D., "Information-Centric Networking (ICN) Research Challenges", RFC 7927, July 2016.
- [20] Thomos, N. and P. Frossard, "Toward one Symbol Network Coding Vectors", IEEE Communications letters, vol. 16, no. 11, November 2012.
- [21] Lucani, D., Pedersen, M., Heide, J., and F. Fitzek, "Fulcrum Network Codes: A Code for Fluid Allocation of Complexity", available at <http://arxiv.org/abs/1404.6620>, April 2014.
- [22] Perino, D. and M. Varvello, "A reality check for content centric networking", Proc. SIGCOMM Workshop on Information-centric networking (ICN'11), ACM, August 2011.
- [23] Wu, Q., Li, Z., Tyson, G., Uhlig, S., Kaafar, M., and G. Xie, "Privacy-Aware Multipath Video Caching for Content-Centric Networks", IEEE Journal of Selected Area (JSAC) vol. 38, no. 8, June 2016.
- [24] Wu, Y., Chou, P., and K. Jain, "A comparison of network coding and tree packing", Proc. ISIT, IEEE, June 2004.
- [25] Podlipnig, S. and L. Osz, "A Survey of Web Cache Replacement Strategies", Proc. ACM Computing Surveys vol. 35, no. 4, December 2003.
- [26] Rossini, G. and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies", Elsevier Computer Communication, vol.36, no. 7, April 2013.

- [27] Chai, W., He, D., Psaras, I., and G. Pavlou, "Cache Less for More in Information-centric Networks", Journal Computer Communications, vol. 37. no. 7, April 2013.
- [28] Carofiglio, G., Muscariello, L., Papalini, M., Rozhnova, N., and X. Zeng, "Leveraging ICN In-network Control for Loss Detection and Recovery in Wireless Mobile networks", Proc. ICN ACM, September 2016.
- [29] Ali, M. and U. Niesen, "Coding for Caching: Fundamental Limits and Practical Challenges", IEEE Communications Magazine vol. 54, no. 8, August 2016.
- [30] Koetter, R. and F. Kschischang, "An algebraic approach to network coding", IEEE Trans. Netw. vol.11, no.5, October 2008.
- [31] Tournoux, P., Lochin, E., Lacan, J., Bouabdallah, A., and V. Roca, "On-the-Fly Erasure Coding for Real-Time Video Applications", IEEE Trans. Multimed. vol.13, no.4, August 2011.

Authors' Addresses

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

Email: matsuzono@nict.go.jp

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

Email: asaeda@nict.go.jp

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

Email: cedric.westphal@huawei.com

NWCRG
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

I. Swett
Google
M. Montpetit
Triangle Video
V. Roca
INRIA
October 30, 2017

Network Layer Coding for QUIC: Requirements
draft-quic-coding-00

Abstract

This document presents the motivation and requirements for the use of Network Level Packet Erasure Coding to improve the performance of the QUIC protocol that is proposed a new transport protocol. The document does not specify a specific code but lists the salient features that a code should have in order to deal with know loss patterns on QUIC paths.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology	2
2. Introduction	2
3. QUIC Background	3
4. Motivation	3
5. Architecture	4
6. Use-cases	5
7. Requirements	5
8. Next Steps	5
9. IANA Considerations	5
10. Security Considerations	5
11. References	6
11.1. Normative References	6
11.2. Informative References	6
Appendix A. Revision information	7
Authors' Addresses	7

1. 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 [RFC2119].

In addition, while most of the the terminology in this document is conform to the taxonomy presented in [[NC-Taxonomy]] for clarity and comparison with existing QUIC documents we continue to use the word packet to indication the entity that will be encoded vs. symbol in the taxonomy document.

NOTE: while using drafts in references is not compliant with IETF/IRTF rules they will be replaced by RFCs as they become available.

2. Introduction

The QUIC (Quick UDP-based Internet Connection) protocol is currently being proposed as new transport protocol than multiplexes connections over UDP. The major elements have been defined and are being implemented by the QUIC IETF working group [QUIC-WG] including wire format, connection establishment, stream multiplexing, stream and connection-level flow control, and data encryption [numerous draft references]. This document addresses an outstanding element of the QUIC protocol, namely how to account and correct for packet losses at

the network layer that will have a very negative impact on transport delay, throughput and reliability. This document presents the salient features and requirements for a network coding (NC) protocol to provide the QUIC packet loss recovery it requires. NC provides a structured, algebraic mechanism to recover lost packets based on a vast heritage of Forward Error Correction (FEC) and has shown better performance of packet recovery than XORs or repetition codes to deal with the losses in the Internet. The Network Coding taxonomy document [[NC-Taxonomy]] contains an overview of top NC concepts. Note: we need a small NC draft that explains how it works.

3. QUIC Background

This section will be completed in a future version. For the needs of the current document we need to know that the QUIC packet format contains a unique packet identifier (ID) and a connection ID. Details will be obtained from [[QUIC-Connect]] and [[QUIC-Trans]]

4. Motivation

The QUIC protocol from its early implementations, wanted to address packet losses in the Internet as they can greatly impact protocol performance and impact the performance congestion control mechanisms [[QUIC-Loss]]. For example TCP goodput goes below 20% with 3% loss [are there any other references besides the famous Mathis curve?]. In this section we review the motivations behind the use of a network coding approach to reduce the impact of packet losses on QUIC. It is important to note that we limit the sources of the losses to IP layer and above and losses in lower layers are addressed by other standardization organization.

It is known (is it?) that the main sources of losses in the Internet include (but are not limited to):

- o Queuing losses across multiple flows
- o Intermittent timeouts
- o Connection losses
- o Residual physical or MAC layer losses
- o Misrouting
- o other?

The main feature of all the patterns associated with the loss events above is the fact that losses appear in clusters (burst or correlated losses). Hence they are not the 'random losses' that can be recovered by non structured mechanisms like XOR or repetitions codes even with high overhead or simple block codes with fixed window sizes. Hence because of the correlated losses, the first requirement for a good code for QUIC is one that allows variable window sizes

that allow to vary with the size of the burst. That will be better suited to recover the losses with statistically approximately the same overhead as the average packet loss without limitations on the loss pattern.

5. Architecture

In order to define the potential NC solution, a detailed architecture is necessary. Hence, the main QUIC/NC architecture topics to be addressed includes the following (each will become a subsection in the future).

- o Type of connection: while unicast and/or multicast/broadcast communications are possible over QUIC it is assumed that an initial implementation will be limited to unicast.
- o Addressing single source flow or multiple flows: at the the coding level, if there is a multiplexing on top of the coding level, already managed by QUIC machinery, it may be totally transparent. We can assume that individual packets and connections can be individually identified.
- o Use of feedback: since the code will need to deal with correlated losses can it benefit from feedback to manage the window as opposed to a fully unidirectional source-destination mechanism. This will allow not to lose any packet part of a current generation before the loss burst ends (we need a reference on window growth and maximum size)
- o Minimization of latency: Latency is key for the solution design. This includes reducing extra delay due to encoding/decoding at the ingress, egress and intermediate nodes (middleboxes) especially on delay sensitive paths. At the same time long bandwidth-delay product networks coding should reduce the overall end-to-end delay experienced by an application significantly by minimizing the effect of packet losses and retransmission on TCP congestion control and throughput.
- o Throughput aspects: it is expected that the QUIC flows will include high throughput flows, very low throughput flows and mixed sizes flows.
- o Interactions with other functionalities: Interactions with congestion control and encryption will also be key. Directions will be taken from [[QUIC-Loss]] and [[QUIC-TLS]] and other relevant documents
- o Code changes and future proofing: any protocol designed within QUIC should be able to maybe use more than one code to change codes easily by without major impact this is to address different network conditions or improve performance if a new code was to be developed. It is assumed that the code remain the same for the full QUIC session lifetime but that within a session at least for

the beginning it should be possible to turn off the coding to prevent catastrophic congestion collapse for example.

6. Use-cases

Note: this will be detailed in the next version of the document

7. Requirements

The initial requirements for the QUIC NC are presented below. This list will help to chose the best solution amongst existing codes.

Requirements:

- o Simplicity/low complexity: both encoding and decoding operations should be simple and ass little complexity to the QUIC operations; the use of systematic coding will be encouraged.
- o Low overhead: the NC overhead to compensate for all losses should be as close as possible to the average loss on the path as to not create additional congestion condition.
- o In network coding: there should be ways to create additional coded symbols inside the network either directly or via partial or full decoding.
- o Multipath: there should be ways to take advantage of multipath communications for example to send packets and coded symbols on different paths to reduce delay and overhead on some delay or loss sensitive paths.
- o Licensing/IPR: the solution should be license/patent free.

8. Next Steps

Besides adding the sections missing in the document based on future discussion it is proposed to define a strawman architecture based on existing codes and using the standard APIs being developed in the RG.

9. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

10. Security Considerations

Security: While NC will not impact security in itself it will be important to verify how NC interacts with current encryption used in QUIC and presented in [[QUIC-TLS]].

11. References

11.1. Normative References

- [NC-Taxonomy]
Abramson, B. and et. al., "Network Coding Taxonomy", Internet-draft draft-irtf-nwcrgr-network-coding-taxonomy-05.txt, July 2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.

11.2. Informative References

- [QUIC-Connect]
Roskind, J., "QUIC: Quick UDP Internet Connections", URL https://docs.google.com/document/d/1RNHkx_VvKWYWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/preview#.
- [QUIC-Loss]
Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", Internet-draft draft-iyengar-quic-loss-recovery-06.txt, September 2017.
- [QUIC-Overview]
"QUIC Overview", URL <https://datatracker.ietf.org/wg/quic/about/>.
- [QUIC-TLS]
Thomson, M. and R. Harrison, "Using Transport Layer Security (TLS) to Secure QUIC", Internet-draft -06.txt, September 2017.
- [QUIC-Trans]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Internet-draft draft-ietf-quic-transport-06.txt, September 2017.

Appendix A. Revision information

XXX RFC-Ed please remove this section prior to publication.

Authors' Addresses

Ian Swett
Google
Cambridge, MA
USA

EMail: ianswett@google.com

Marie-Jose Montpetit
Triangle Video
Boston, MA
USA

EMail: marie@mjmontpetit.com

Vincent Roca
INRIA
Grenoble
France

EMail: vincent.roca@inria.fr

NWCRG
Internet-Draft
Intended status: Informational
Expires: April 29, 2018

V. Roca (Ed.)
INRIA
J. Detchart
ISAE - Supaero
C. Adjih
INRIA
M. Pedersen
Aalborg University
I. Swett
Google
October 26, 2017

Generic Application Programming Interface (API) for Window-Based Codes
draft-roca-nwcrg-generic-fec-api-00

Abstract

This document introduces a generic Application Programming Interface (API) for window-based FEC codes. 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 (e.g., signalling) that are the responsibility of the application making use of the codec. A goal of this document is to pave the way for a future open-source implementation of such codes.

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 April 29, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Definitions and Abbreviations	3
3. Existing APIs	3
3.1. Morten API proposal	3
3.2. Jonathan API proposal	3
3.3. Cedric API proposal	8
3.4. Ian API proposal	9
3.5. Vincent API proposal	9
3.5.1. General	9
3.5.2. Session Management	10
3.5.3. Callback Functions	12
3.5.4. Coding window functions	13
3.5.5. Coding coefficients functions	13
3.5.6. Encoder specific functions	15
3.5.7. Decoder specific functions	15
4. Security Considerations	17
5. IANA Considerations	17
6. Acknowledgments	17
7. Normative References	17
Authors' Addresses	17

1. Introduction

This document introduces a generic Application Programming Interface (API) for window-based FEC codes. This API is meant to be usable by any window-based 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 (e.g., signalling) that are the responsibility of the application making use of the codec.

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

This API must be compatible with:

- o MDS and non-MDS codes;
- o Fixed rate and rateless codes;
- o Codes restricted to end-to-end use-cases as well as codes compatible with in-network re-encoding use-cases;

Since this is a usage consideration, 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.

Last but not least, having a common generic API is future-proof. Whatever may appear in the future can be more easily integrated into existing software thanks to a common API.

A few words about block FEC codes and why we do not try to encompass them in this API...

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. Existing APIs

Editor's comment: for the moment, what follows is meant to list the various proposals in order to be able to compare and agree on what should be done.

3.1. Morten API proposal

3.2. Jonathan API proposal

```
<CODE BEGINS>
/** a status for function calls */
typedef enum {
    STATUS_OK,
    STATUS_ERROR,
    /* ... */
}
```

```
} status_t;

/** defines the galois field used (at least the size, maybe we need to separate
the implementations (Lookup Tables or Xor-based)) */
typedef enum {
    GF_16,
    GF_64,
    GF_256
} galois_field_t;

/*
 * coding coefficient generators: specifies the algorithm used to generate the c
oefficients for the linear combinations
 *
 * NOTE: the choice of the finite field could done here (RLC_GF256, RLC_GF_16, .
..) rather than using a different structure
 */
typedef enum {
    RLC,
    VDM
    /* ... */
} coding_coefficients_generator_identifier_t;

/**
 **
 ** encoder side
 **
 **/

/**
 * NOTE for the callbacks in the sw_encoder: this is a proposition, we could als
o use 2 structures (source_t and repair_t) to avoid sending too many parameters
in the callbacks.
 */

/**
 * context: a context as a generic pointer (defined by the application if needed
and given in sw_encoder_set_callbacks)
 * src: the source data unit to consider
 * src_id: the id of the source unit set by the encoder
 * src_sz: the size in bytes of the data unit
 */
typedef void (*sw_encoder_callback_source_ready)(void *context, void* src, uint32
_t src_id, size_t src_sz);

/**
 * context: a context as a generic pointer (defined by the application if needed
and given in sw_encoder_set_callbacks)
 * rep: the repair data unit generated by the encoder
 * rep_id: the id of the repair unit given by the encoder

```

```

* rep_sz: the size in bytes of the repair data unit
* src_ids: the array of the source unit ids used to generate the repair unit
* src_coefs: the coefficients used for each source units to generate the repair
unit
* nb_src_in: number of src units in the linear combination (repair unit) (also
the number of elements of src_ids and src_coefs)
**/
typedef void (*sw_encoder_callback_repair_ready)(void *context, void* rep, uint32
_t rep_id, size_t rep_sz, uint32_t* src_ids, uint8_t *src_coefs, size_t nb_src_i
n);

/** a structure containing all we need to encode */
typedef struct sw_encoder sw_encoder_t;

/**
 * @brief Init a sliding window encoder by giving a galois field size, a coeffic
ient generator function, and the maximum size of the window
 * @param galois_field The size of the galois field used to
create the coefficients and to encode.
 * @param ccgi The function used to generate
the coefficients (depends on the gf_size)
 * @param max_wnd_size set the maximum of source units t con
sider inside the coding window (the oldest units will be destroyed)
 * @return a sw_encoder_t structure.
**/
sw_encoder_t* sw_encoder_init(galois_field_t galois_field, coding_coefficients_g
enerator_identifier_t ccgi, uint32_t max_wnd_size);

/**
 * @brief Set the callbacks to get the encoded (repair) data
 * @param encoder The encoder initialized
 * @param context A generic context defined by the ap
plication (will be given in the callback)
 * @param src_callback The function to be called when a sour
ce data unit has been processed by the encoder
 * @param rep_callback The function to be called when a repai
r data unit has been generated
**/
status_t sw_encoder_set_callbacks(sw_encoder_t* encoder, void* context, sw_encod
er_callback_source_ready src_callback, sw_encoder_callback_repair_ready rep_call
back);

/**
 * @brief Gives a source data unit and its id to the encoder
 * @param encoder The encoder structure.
 * @param src The data to add
 * @param sz The size in bytes of the data unit
**/
status_t sw_encoder_add_source(sw_encoder_t* encoder, void* src, size_t sz);

/**
 * @brief Removes the corresponding source data unit from the encoding window by
giving and id
 * @param encoder The encoder structure
 * @param id The id of the data unit
 * @return STATUS_OK if the data unit has been found and rem
oved, STATUS_ERROR if the data unit doesn't exist
**/

```

```
status_t sw_encoder_remove_source(sw_encoder_t* encoder, uint32_t id);
```

```
/**
 * @brief generates a repair data unit and calls the corresponding callback.
 * @param encoder          The encoder structure.
 */
status_t sw_encoder_generate_repair(sw_encoder_t* encoder);

/* ... */
status_t sw_encoder_set_control_parameter(sw_encoder_t* encoder, uint32_t type,
void* value, uint32_t length);

/* ... */
status_t sw_encoder_get_control_parameter(sw_encoder_t* encoder, uint32_t type,
void* value, uint32_t length);

/**
 * @brief Release an encoder structure
 * @param encoder          The encoder structure
 */
status_t sw_encoder_release(sw_encoder_t* encoder);

/**
 **
 ** decoder side
 **
 **/

/**
 * NOTE for the callback in the sw_decoder: this is a proposition, we could also
 * use an opaque structure (source_t) to avoid sending too many parameters in the
 * callback.
 */
/**
 * context: a context as a generic pointer (defined by the application if needed
 * and given in sw_encoder_set_callbacks)
 * src: the source data unit to consider
 * src_id: the id of the source unit used in the decoder
 * src_sz: the size in bytes of the data unit
 */
typedef void (*sw_decoder_callback_source_ready)(void *context, void* src, uint32
_t src_id, size_t sz);

/** a structure containing all we need to decode */
typedef struct sw_decoder sw_decoder_t;

/**
 * @brief Init a sliding window decoder by giving a galois field size, a coeffic
 * ient generator function, and the maximum size of the window
 * @param galois_field          The size of the galois field used to
 * create the coefficients and to encode.
```



```

* @param ccgi                                The function used to generate
the coefficients (depends on the gf_size)
* @return a sw_decoder_t structure.
**/
sw_decoder_t* sw_decoder_init(galois_field_t galois_field, coding_coefficients_g
enerator_identifier_t ccgi);

/**
* @brief Set the callback to get the encoded (repair) data
* @param decoder                                The decoder initialized
* @param context                                A generic context defined by the ap
plication (will be given in the callback)
* @param callback                                The function to be called
**/
status_t sw_decoder_set_callback_source_ready(sw_decoder_t* decoder, void* conte
xt, sw_decoder_callback_source_ready callback);

/**
* NOTE for the next decode functions: we could also use 2 opaque structures to
represent the source and repair units (source_t or repair_t)
**/

/**
* @brief decode some source data units by giving to the decoder a new source da
ta unit
* @param decoder                                the decoder structure
* @param src                                    the new source data unit
* @param src_id                                the id of the new source data unit
(given by an encoder)
* @param src_sz                                the size in bytes of the new source
data unit
**/
status_t sw_decoder_decode_with_source(sw_decoder_t* decoder, void* src, uint32_
t src_id, size_t src_sz);

/**
* @brief decode some source data units by giving to the decoder a new repair da
ta unit
* @param decoder                                the decoder structure
* @param rep                                    the new repair data unit
* @param rep_id                                the id of the new repair data unit
(given by an encoder)
* @param rep_sz                                the size in bytes of the new repair
data unit
* @param src_ids:                              the array of the source unit ids
used to generate this repair unit
* @param src_coefs:                            the coefficients used for each source u
nits to generate this repair unit
* @param nb_src_in:                            number of src units in the linear combi
nation (repair unit) (also the number of elements of src_ids and src_coefs)
**/
status_t sw_decoder_decode_with_repair(sw_decoder_t* decoder, void* rep, uint32_
t rep_id, size_t rep_sz, uint32_t* src_ids, uint8_t * src_coefs, size_t nb_src_i
n);

/* ... */
status_t sw_decoder_set_control_parameter(sw_decoder_t* decoder, uint32_t type,
void* value, uint32_t length);

```

/* ... */

Roca (Ed.), et al.

Expires April 29, 2018

[Page 7]

```
status_t sw_decoder_get_control_parameter(sw_decoder_t* decoder, uint32_t type,
void* value, uint32_t length);
```

```
/**
 * @brief Release a decoder structure
 * @param decoder      The decoder structure to release
 */
status_t sw_decoder_release(sw_decoder_t* decoder);
<CODE ENDS>
```

Jonathan API proposal

3.3. Cedric API proposal

For DRAGONCAST/DragonNet/GardiNet:

- o an API could be globally pretty similar;
- o there is a maintained set of symbols of the "codec" where online Gaussian Elimination is performed. But this same set, is used to also re-code packets for generation. For this to work, one uses as pivot the highest index (instead of the lowest in standard RREF), in order to avoid adding symbols with higher indices in the decoding process.
- o another set of differences would be that the protocol has more control over the coding process than our current codec proposal. The reason is that DRAGONCAST (re)codes for several neighbors, and in such scenario, there is no "obvious" decision that can be made, for instance:
 - * which source symbols (indices) should be present in a generated packet: -> tradeoff: helping the maximum number of nodes (emphasis on "new" undecoded source symbol indices) -vs- helping the neighbor which is the most late in the decoding process (emphasis on "old" source symbols)
 - * which symbols should be kept in the decoding process (or dropped): -> tradeoff: helping coding by keeping old symbols (be able to generate symbols for late neighbors) vs keeping up with decoding (and never throwing away a new symbol with high indices). (I. Amdouni discussed such issues in <https://tools.ietf.org/html/draft-amdouni-nwcrg-cisew-00> for instance).
- o In the current implementation, the packet generation process is done in the protocol which directly "peeks" in the set of symbols in the codec, and creates a linear combination with the ones that suits it.
- o Technically the callbacks from the "codec" are:
 - * notification of a source symbol is decoded;

- * notification that the set of symbols is full (protocol can remove symbols it sees fits, especially decoded symbols);
- * for the "over-the-air" reflashing application, the codec can ask the protocol if an already removed source symbol is available (on the assumption that it has been written somewhere else);

3.4. Ian API proposal

3.5. Vincent API proposal

3.5.1. General

<CODE BEGINS>

```
/**
 * The fec_codec_id_t enum identifies the FEC code/codec being used.
 * Since a given fec_codec_id can be used by one or several FEC schemes (that specify
 * both the codes and way of using these codes), it is distinct from the FEC Encoding
 * ID.
 */
typedef enum {
    CODEC_NIL = 0,
    CODEC_RLC
} codec_id_t;

/**
 * Function return value, indicating whether the function call succeeded or not.
 * In case of failure, the detailed error type is returned in a global variable,
 * of_errno (see of_errno.h).
 *
 * STATUS_OK = 0          Success
 * STATUS_FAILURE,       Failure. The function called did not succeed to perform
 *                        its task, however this is not an error. This can happen
 *                        for instance when decoding did not succeed (which is a
 *                        valid output).
 * STATUS_ERROR,         Generic error type. The caller is expected to be able
 *                        to call the library in the future after having corrected
 *                        the error cause.
 * STATUS_FATAL_ERROR    Fatal error. The caller is expected to stop using this
 *                        codec instance immediately (it replaces an exit() system
 *                        call).
 */
typedef enum {
    STATUS_OK = 0,
    STATUS_FAILURE,
    STATUS_ERROR,
    STATUS_FATAL_ERROR
}
```

```

} status_t;

/**
 * Throughout the API, a pointer to this structure is used as an identifier of the current
 * codec instance, also known as "session".
 *
 * This generic structure is meant to be extended by each codec and new pieces of
 * information that are specific to each codec be specified there. However, all the codec specific
 * structures MUST begin the same entries as the ones provided in this generic structure, otherwise
 * hazardous behaviors may happen.
 */
typedef struct session {
    codec_id_t    codec_id;
    codec_type_t  codec_type;
} session_t;

/**
 * Generic FEC parameter structure used by set_fec_parameters().
 *
 * This generic structure is meant to be extended by each codec and new pieces of
 * information that are specific to each codec be specified there. However, all the codec specific
 * structures MUST begin the same entries as the ones provided in this generic structure, otherwise
 * hazardous behaviors may happen.
 */
typedef struct {
    /** SENDER and RECEIVER: maximum number of source symbols used for any re
    pair symbol. */
    UINT32      coding_window_max_size;

    /** RECEIVER only: maximum number of source symbols kept in current line
    ar
    system. If the linear system grows above this limit, older source sym
    bols
    in excess are removed and the application callback called if set. Thi
    s
    value MUST be larger than the coding_window_max_size. */
    UINT32      linear_system_max_size;
    UINT32      encoding_symbol_length;
} parameters_t;
<CODE ENDS>

```

Vincent API proposal

3.5.2. Session Management

```

<CODE BEGINS>
/**
 * This function allocates and partially initializes a new session structure.
 * Throughout the API, a pointer to this session is used as an identifier of the

```



```

* current codec instance.
*
* @param ses          (IN/OUT) address of the pointer to a session. This pointer is updated
*                      by this function.
*                      In case of success, it points to a session structure allocated by the
*                      library. In case of failure it points to NULL.
* @param codec_id     identifies the FEC code/codec being used.
* @param codec_type    indicates if this is a coder or a decoder.
* @param verbosity    set the verbosity level
* @return             Completion status. The ses pointer is updated according to the success return status.
*/
status_t create_codec_instance (session_t** ses,
                               codec_id_t  codec_id,
                               codec_type_t codec_type,
                               uint32_t     verbosity);

/**
 * This function releases all the internal resources used by this FEC codec instance.
 * None of the source symbol buffers will be free'd by this function, even those decoded by
 * the library if any, regardless of whether a callback has been registered or not. It's the
 * responsibility of the caller to free them.
 *
 * @param ses          (IN) Pointer to the session.
 * @return             Completion status
 */
status_t release_codec_instance (session_t* ses);

/**
 * Second step of the initialization, where the application specifies code(c) specific parameters.
 *
 * At a receiver, the parameters can be extracted from the FEC OTI that is usually communicated
 * to the receiver by either an in-band mechanism or an out-of-band mechanism, or set statically
 * for a specific use-case.
 *
 * @param ses          (IN) Pointer to the session.
 * @param params       (IN) pointer to a structure containing the FEC parameters associated to
 *                      a specific FEC codec.
 * @return             Completion status.
 */
status_t set_fec_parameters (session_t* ses,
                             parameters_t* params);

/**
 * This function sets a FEC scheme/FEC codec specific control parameter,
 * using a type/value method.

```

```

*
* @param ses          (IN) Pointer to the session.
* @param type         (IN) Type of parameter. This type is FEC codec ID specif
ic.
* @param value        (IN) Pointer to the value of the parameter. The type of
the object pointed
*                      is FEC codec ID specific.
* @param length       (IN) length of pointer value
* @return             Completion status.
*/
status_t      set_control_parameter (session_t*   ses,
                                   UINT32         type,
                                   void*          value,
                                   UINT32         length);

/**
* This function gets a FEC scheme/FEC codec specific control parameter,
* using a type/value/length method.
*
* @param ses          (IN) Pointer to the session.
* @param type         (IN) Type of parameter. This type is FEC codec ID specif
ic.
* @param value        (IN/OUT) Pointer to the value of the parameter. The type
of the object
*                      pointed is FEC codec ID specific. This function updates
the value object
*                      accordingly. The application, who knows the FEC codec ID
, is responsible
*                      to allocating the appropriate object pointed by the valu
e pointer.
* @param length       (IN) length of pointer value
* @return             Completion status.
*/
status_t      get_control_parameter (session_t*   ses,
                                   UINT32         type,
                                   void*          value,
                                   UINT32         length);
<CODE ENDS>

```

Vincent API proposal

3.5.3. Callback Functions

```

<CODE BEGINS>
/**
* Set the various callback functions for this session.
* All the callback functions require an opaque context parameter, that must be
* initialized accordingly by the application, since it is application specific.
*
* @param ses          (IN) Pointer to the session.
*
* @param decoded_source_symbol_callback
*                      (IN) Pointer to the function, within the application, th
at
*                      needs to be called each time a source symbol is decoded.

```



```

*
* @param available_source_symbol_callback
*           (IN) Pointer to the function, within the application, th
at
*           needs to be called each time a source symbol is decoded
and
*           all computations performed (i.e., the buffer does contai
n the
*           symbol value).
*
* @param source_symbol_removed_from_coding_window_callback
*           (IN) Pointer to the function, within the application, th
at
*           needs to be called each time a source symbol is removed
from
*           the left side of the coding window, at a SENDER because
this
*           window has slided to the right, or at a RECEIVER because
this
*           old source symbol is now forgotten.
*
* @param context_4_callback
*           (IN) Pointer to the application-specific context that wi
ll be
*           passed to the callback function (if any). This context i
s not
*           interpreted by this function.
*
* @return   Completion status.
*/

```

```

status_t      set_callback_functions (of_session_t*      ses,
void* (*decoded_source_symbol_callback) (void *context,
                                         UINT32      size,          /* size
of decoded source symbol */
                                         UINT32      esi),          /* enco
ding symbol ID */
void (*available_source_symbol_callback) (void      *context,
void      *new_symbol_buf, /* symb
ol buffer */
                                         UINT32      size,          /* size
of decoded source symbol */
                                         UINT32      esi),          /* enco
ding symbol ID */
void (*source_symbol_removed_from_coding_window_callback)
                                         (void      *context,
                                         UINT32      old_symbol_esi),
void*          context_4_callback);
<CODE ENDS>

```

Vincent API proposal

3.5.4. Coding window functions

TBD

3.5.5. Coding coefficients functions

<CODE BEGINS>

```

/**
* SENDER:   this function specifies the coding coefficients chosen by the applic
ation if this is the way the codec

```



```

*      works. This function MUST be called before calling build_repair_symbol().
* RECEIVER: communicate the coding coefficients associated to a repair symbol and
*      carried in the packet header.
*      This function MUST be called before calling decode_with_new_repair_symbol
*      ().
*
* @param ses
* @param coding_coefs_tab      (IN) table of coding coefficients to be associated
*      to each of the source symbols
*      currently in the coding 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                      Completion status.
*/
status_t      set_coding_coefficients_tab (session_t*      ses,
                                           void*            coding_coefs_tab,
                                           UINT32           nb_coefs_in_tab);

/**
* SENDER:   this function enables the application to retrieve the set of coding
*      coefficients generated and used by
*      build_repair_symbol().
* RECEIVER: never used.
*
* @param ses
* @param coding_coefs_tab      (IN/OUT) pointer of a table of coding coefficients
*      to be associated to each of the
*      source symbols currently in the coding 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. Upon
* return of this function, this
*      table is allocated and filled with each coefficient
* value.
* @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 number is initialized with the actual number
* of entries in the coeffs_tab[].
* @return                      Completion status (OF_STATUS_OK, FAILURE, ERROR or
*      FATAL_ERROR).
*/
status_t      get_coding_coefficients_tab (session_t*      ses,
                                           void**          coding_coefs_tab,
                                           UINT32*         nb_coefs_in_tab);

/**
* The coding coefficients may be generated in a deterministic manner, (e.g., through
* the use of a PRNG and the
* repair symbol ESI used as a seed). This is the case with RLC codes.
*
* SENDER:   generate all coefficients. This function MUST be called before calling
* build_repair_symbol().
* RECEIVER: generate all coefficients. This function MUST be called before calling
* decode_with_new_repair_symbol().
*
* @param ses

```

```
* @param params      (IN) pointer to a codec specific structure contain-
ning the required parameters.
*
*                   These parameters can include a repair symbol ESI
or key among other things.
* @return             Completion status.
*/
```

```
status_t      generate_coding_coefficients (session_t*    ses,  
                                           void*          params);  
<CODE ENDS>
```

Vincent API proposal

3.5.6. Encoder specific functions

```
<CODE BEGINS>  
/**  
 * Create a single repair symbol, i.e. perform an encoding.  
 * This function requires that the application has previously set the coding win  
 * dow and if needed the coding coefficients  
 * appropriately. After that, the application can call this function.  
 *  
 * @param ses  
 * @param new_repair_symbol_buf (IN) The pointer to the buffer for the repair sy  
 * mbol to build can either point to a buffer  
 * allocated by the application, or let to NULL mea  
 * ning that this function will allocate  
 * memory.  
 * @return Completion status.  
 */  
status_t      ccod_build_repair_symbol (session_t*    ses,  
                                         void*          new_repair_symbol_buf);  
<CODE ENDS>
```

Vincent API proposal

3.5.7. Decoder specific functions

<CODE BEGINS>

```

/**
 * 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 fu
 * nctions.
 *
 * This function usually returns OF_STATUS_OK, regardless of whether this new sy
 * mbol enabled the
 * decoding of one or several source symbols, unless an error occurred. This func
 * tion cannot return
 * OF_STATUS_FAILURE.
 *
 * @param ses
 * @param new_src_symbol_buf (IN) Pointer to the new source symbol now availa
 * ble (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_or_key (IN) encoding symbol ID of the new source symbol
 * or key if there is no notion of ESI.
 * @return Completion status.
 */
status_t decode_with_new_source_symbol (session_t* ses,
                                       void* const new_src_symbol_buf,
                                       UINT32 new_symbol esi_or_ke
y);

```

```

/**
 * 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 fu
 * nctions.
 *
 * This function requires that the application has previously set the coding win
 * dow and the coding coefficients appropriately.
 * After that, the application can call this function. The
 * application keeps a full control of the repair symbol buffer, i.e., the appli
 * cation is in charge
 * of freeing this buffer as soon as it believes appropriate to do so (a copy is
 * kept by the codec).
 *
 * This function usually returns OF_STATUS_OK, regardless of whether this new sy
 * mbol enabled the
 * decoding of one or several source symbols, unless an error occurred. This fun
 * ction cannot return
 * OF_STATUS_FAILURE.
 *
 * @param ses
 * @param new_repair_symbol_buf (IN) Pointer to the new repair symbol now availa
 * ble (i.e. a new symbol received by
 * the application or a decoded symbol in case of a
 * recursive call if it makes sense).
 * @return Completion status.
 */
status_t decode_with_new_repair_symbol (session_t* ses,
                                       void* const new_repair_symbol_buf)
;
<CODE ENDS>

```


4. Security Considerations

TBD

5. IANA Considerations

N/A.

6. Acknowledgments

The authors would like to thank TBD.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

Authors' Addresses

Vincent Roca
INRIA
Grenoble
France

EMail: vincent.roca@inria.fr

Jonathan Detchart
ISAE - Supaero
France

EMail: jonathan.detchart@isae-supaero.fr

Cedric Adjih
INRIA
France

EMail: cedric.adjih@inria.fr

Morten V. Pedersen
Aalborg University
Denmark

EMail: mvp@es.aau.dk

Ian Swett
Google
USA

EMail: ianswett@google.com

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on May 20, 2020.

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

2. Definitions and Abbreviations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses the following definitions and abbreviations:

XXX

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

This generic FEC API is meant to be used with:

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

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

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

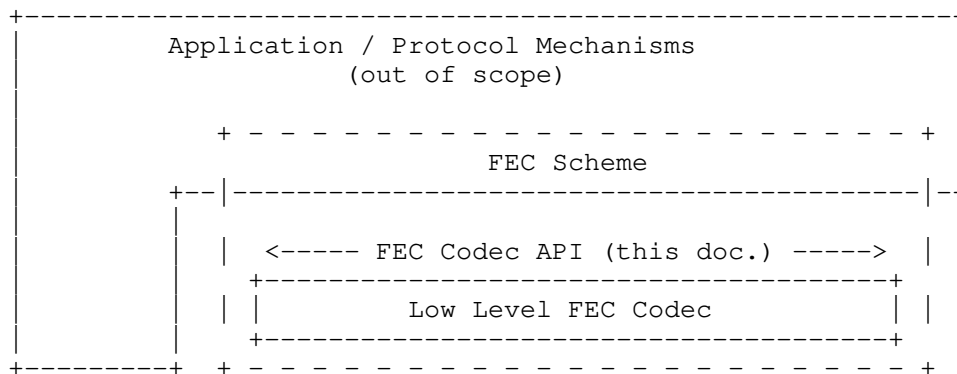


Figure 1: Position of the FEC Codec API with respect to the low level FEC Codec, the FEC Scheme, the protocol and other caller services.

3.1. Mechanisms Considered or Ignored by the API

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

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

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

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

4. Generic API for Sliding Window FEC Codes

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

4.1. General Definitions Common to the Encoder and Decoder

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

About FEC Codepoints:

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

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

<CODE BEGINS>

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

```
        SWIF_STATUS_ERROR
    } swif_status_t;

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

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

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

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

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

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



```

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

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

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

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

```

```

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

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

```

General definitions.

4.2. Encoder

```

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

```

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

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

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

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

```

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

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

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

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

```

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

```

Encoder API proposal

```

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

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

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

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

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

```

Non normative example of internal structure used by an encoder.

4.3. Decoder

<CODE BEGINS>

```

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

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

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

```

```

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

/**

```

```

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

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

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

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

```



```

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

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

```

```

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

```

Decoder API proposal

```

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

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

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

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

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

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

```

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

4.4. Coding Window Functions at an Encoder and Decoder

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

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

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

<CODE BEGINS>

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

swif_status_t    swif_decoder_reset_coding_window (swif_decoder_t*  dec);

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

```

swif_status_t    swif_decoder_add_source_symbol_to_coding_window (
                                swif_decoder_t* dec,
                                esi_t            new_src_symbol_esi);

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

swif_status_t    swif_decoder_remove_source_symbol_from_coding_window (
                                swif_decoder_t* dec,
                                esi_t            old_src_symbol_esi);

<CODE ENDS>

```

Coding Window Functions at an Encoder and Decoder.

4.5. Coding Coefficients Functions at an Encoder and Decoder

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

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

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

<CODE BEGINS>

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

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

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

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

```

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

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

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

```

```

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

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

```

Coding Coefficients Functions at an Encoder and Decoder.

5. Security Considerations

TBD

6. IANA Considerations

This document has no IANA requirement.

7. Acknowledgments

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

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

8.2. Informative References

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

Authors' Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Jonathan Detchart
ISAE - Supaero
France

EMail: jonathan.detchart@isae-supaero.fr

Cedric Adjih
INRIA
France

EMail: cedric.adjih@inria.fr

Morten V. Pedersen
Steinwurf ApS
Denmark

EMail: morten@steinwurf.com