NWCRG
Internet-Draft
Intended status: Experimental
Expires: September 10, 2015

J. Detchart
E. Lochin
J. Lacan
ISAE
V. Roca
INRIA

March 9, 2015

Tetrys, an On-the-Fly Network Coding protocol
draft-detchart-nwcrg-tetrys-01

Abstract

This document describes Tetrys, an On-The-Fly Network Coding (NC) protocol that can be used to transport delay and loss sensitive data over a lossy network. Tetrys can recover from erasures within a RTT-independent delay, thanks to the transmission of coded packets (redundancy). It can be used for both unicast, multicast and anycast communications.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://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 10, 2015.

Copyright Notice

Table of Contents

1. Introduction

This document describes Tetrys, a novel network coding protocol. Network codes were introduced in the early 2000s [AHL-00] to address the limitations of transmission over the Internet (delay, capacity and packet loss). While the use of network codes is fairly recent in the Internet community, the use of application layer erasure codes in the IETF has already been standardized in the RMT [RMT] and the FECFRAME [FECFRAME] working groups. The protocol presented here can be seen as a network coding extension to standards solutions. The current proposal can be considered as a combination of network erasure coding and feedback mechanisms [Tetrys].

The main innovation of the Tetrys protocol is in the generation of coded packets from an elastic encoding window, the size of which is periodically updated with the receiver's feedback. This update is done in such a way that any source packets coming from an input flow is included in the encoding window as long as it is not acknowledged or the encoding window did not reach a limit. This mechanism allows for losses on both the forward and return paths and in particular is resilient to acknowledgement losses.

With Tetrys, a coded packet is a linear combination of the data source packets belonging to the coding window over a finite field. The choice of the finite field of the coefficients is a trade-off between the best performance (with non-binary coefficients) and the system constraints (binary codes in an energy constrained environment) and is driven by the application.

Thanks to the elastic encoding window, the coded packets are built on-the-fly, by using an algorithm or a function to choose the coefficients. The redundancy ratio can be dynamically adjusted, and the coefficients can be generated in different ways along a transmission. This allows to reduce the bandwidth used, compared to FEC block codes, and the decoding delay.

1.1. Requirements Notation

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

2. Definitions, Notations and Abbreviations

The terminology used in this document is presented below. It is aligned with the FECFRAME terminology as well as with recent activities in the Network Coding Research Group.

Source symbol: a symbol that has to be transmitted between the ingress and egress of the network.

Coded symbol: a linear combination over a finite field of a set of source symbols.

Source symbol ID: a sequence number to identify the source symbols.

Coded symbol ID: a sequence number to identify the coded symbols.

Encoding vector: a set of the encoding coefficients and input symbol IDs. One or both sets can be null.

Source packet: a source packet contains a source symbol with its associated IDs.

Coded packet: a coded packet contains a coded symbol, the coded symbol's ID and encoding vector.

Input symbol: a symbol at the input of the Tetrys Encoding Building Block.

Output symbol: a symbol generated by the Tetrys Encoding Building Block. For a non systematic mode, all output symbols are coded symbols. For a systematic mode, output symbols can be the input symbols and a number of coded symbols that are linear combinations of the input symbols + the encoding vectors.

Feedback packet: a feedback packet is a packet containing information about the decoded or received source symbols. It can also bring additional information about the Packet Error Rate or the number of various packets in the receiver decoding window

Elastic Encoding Window: an encoder-side buffer that stores all the non-acknowledged source packets of the input flow that are part of the coding process.

Coding Coefficient Generator Identifier: a unique identifier to define a function or an algorithm allowing to generate the coefficients used to compute the coded packets.

Code rate: Define the rate of generating and sending the redundancy.

3. Architecture

-- Editor's note: The architecture used in this document should be aligned with the future NC Architecture document [NWCRG-ARCH]. --

3.1. Use Cases

Tetrys is well suited, but not limited to the use case where there is a single flow originated by a single source, with intra stream coding that takes place at a single encoding node. Transmission can be over a single path or multiple paths. In addition, the flow can be sent in unicast, multicast, or anycast mode. This is the simplest use-case, that is very much inline with currently proposed scenarios for end-to-end streaming.

3.2. Overview

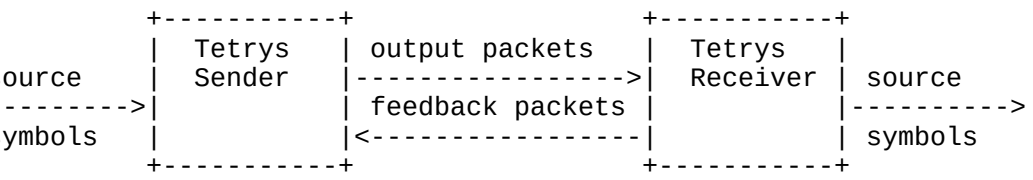


Figure 1: Tetrys Architecture.

The Tetrys protocol features several key functionalities:

- On-the-fly encoding;
- Decoding;
- Signaling, to carry in particular the symbol identifiers in the encoding window and the associated coding coefficients when meaningful, in a manner that was previously used in FEC;
- Feedback management;
- Elastic window management;
- Channel estimation;
- Dynamic adjustment of the code rate and flow control;
- Congestion control management (if appropriate);

  -- Editor’s note: must be discussed --

- Tetrys packet header creation and processing;
- -- Editor’s note: something else? --

These functionalities are provided by several building blocks:

- The Tetrys Building Block: this BB is used during encoding and decoding processes. It must be noted that Tetrys does not mandate a specific building block. Instead any building block compatible with the elastic encoding window feature of Tetrys can be used.
- The Window Management Building Block: this building block is in charge of managing the encoding encoding window at a Tetrys sender.

-- Editor's note: Is it worth moving it in a dedicated BB? To be discussed --

- Other ?

In order to enable future components and services to be added dynamically, Tetrys adds a header extension mechanism, compatible with that of LCT, NORM, FECFRAME [REFS].

4. Packet Format

4.1. Common Header Format

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

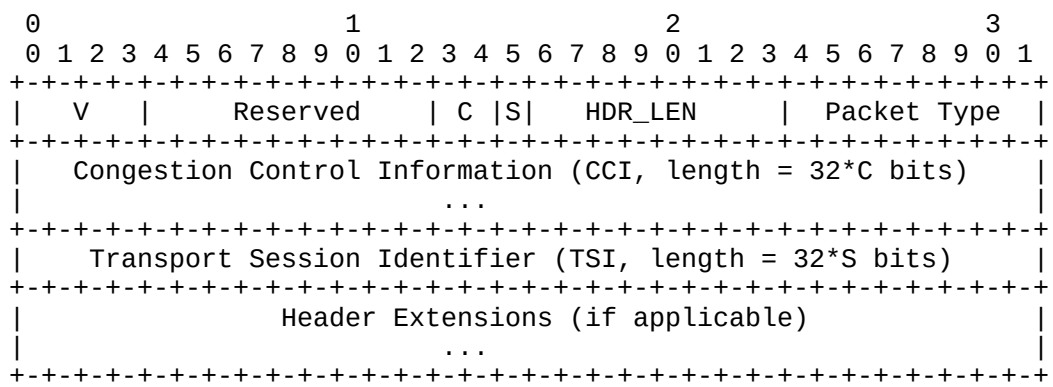


Figure 2: Common Header Format

-- Editor's note: this format inherits from the LCT header format (RFC 5651) with slight modifications. --

- Tetrys version number (V): 4 bits. Indicates the Tetrys version number. The Tetrys version number for this specification is 1.

- Reserved (Resv): 9 bits. These bits are reserved. In this version of the specification, they MUST be set to zero by senders and MUST be ignored by receivers.

- Congestion control flag (C): 2 bits. C=0 indicates the Congestion Control Information (CCI) field is 0 bits in length. C=1 indicates the CCI field is 32 bits in length. C=2 indicates the CCI field is 64 bits in length. C=3 indicates the CCI field is 96 bits in length.

-- Editor's note: version number and congestion control to be discussed --

- Transport Session Identifier flag (S): 1 bit. This is the number of full 32-bit words in the TSI field. The TSI field is 32*S bits in length, i.e., the length is either 0 bits or 32 bits.

- Header length (HDR_LEN): 8 bits. Total length of the Tetrys header in units of 32-bit words. The length of the Tetrys header MUST be a multiple of 32 bits. This field can be used to directly access the portion of the packet beyond the Tetrys header, i.e., to the first other header if it exists, or to the packet payload if it exists and there is no other header, or to the end of the packet if there are no other headers or packet payload.

- Packet Type: 8 bits. Type of packet.

- Congestion Control Information (CCI): 0, 32, 64, or 96 128 bits Used to carry congestion control information. For example, the congestion control information could include layer numbers, logical channel numbers, and sequence numbers. This field is opaque for the purpose of this specification. This field MUST be 0 bits (absent) if C=0. This field MUST be 32 bits if C=1. This field MUST be 64 bits if C=2. This field MUST be 96 bits if C=3.

- Transport Session Identifier (TSI): 0, 16, 32, or 48 bits The TSI uniquely identifies a session among all sessions from a particular sender. The TSI is scoped by the IP address of the sender, and thus the IP address of the sender and the TSI together uniquely identify the session. Although a TSI in conjunction with the IP address of the sender always uniquely identifies a session, whether or not the TSI is included in the Tetrys header depends on what is used as the TSI value. If the underlying transport is UDP, then the 16-bit UDP source port number MAY serve as the TSI for the session. If the TSI value appears multiple times in a packet, then all occurrences MUST be the same value. If there is no underlying TSI provided by the network, transport or any other layer, then the TSI MUST be included in the Tetrys header.

4.1.1. Header Extensions

Header Extensions are used in Tetrys to accommodate optional header fields that are not always used or have variable size. The presence of Header Extensions can be inferred by the Tetrys header length (HDR_LEN). If HDR_LEN is larger than the length of the standard header, then the remaining header space is taken by Header Extension fields.

If present, Header Extensions MUST be processed to ensure that they are recognized before performing any congestion control procedure or otherwise accepting a packet. The default action for unrecognized Header Extensions is to ignore them. This allows the future introduction of backward-compatible enhancements to Tetrys without changing the Tetrys version number. Non-backward-compatible Header Extensions CANNOT be introduced without changing the Tetrys version number.

There are two formats for Header Extension fields, as depicted in Figure Figure 3. The first format is used for variable-length extensions, with Header Extension Type (HET) values between 0 and 127. The second format is used for fixed-length (one 32-bit word) extensions, using HET values from 127 to 255.

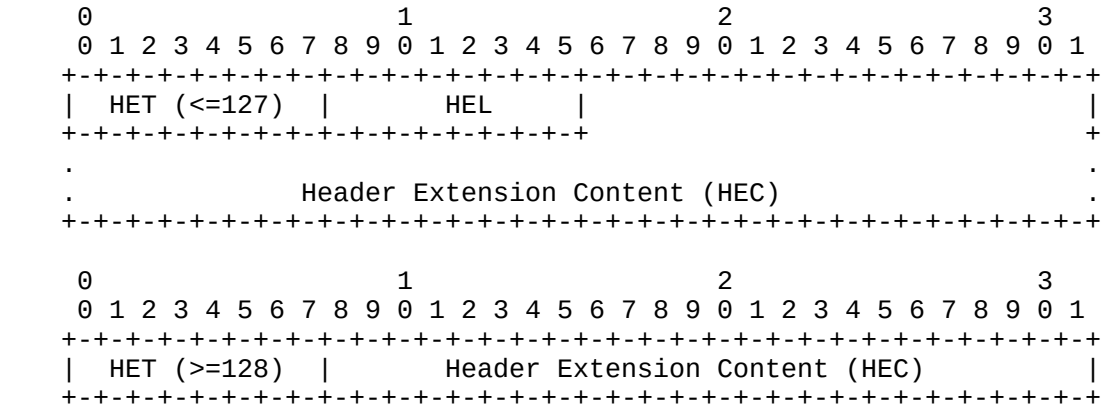


Figure 3: Header Extension Format

- Header Extension Type (HET): 8 bits The type of the Header Extension. This document defines a number of possible types. Additional types may be defined in future versions of this specification. HET values from 0 to 127 are used for variable-length Header Extensions. HET values from 128 to 255 are used for fixed-length 32-bit Header Extensions.

- Header Extension Length (HEL): 8 bits The length of the whole Header Extension field, expressed in multiples of 32-bit words. This field MUST be present for variable-length extensions (HETs between 0 and 127) and MUST NOT be present for fixed-length extensions (HETs between 128 and 255).

- Header Extension Content (HEC): variable length The content of the Header Extension. The format of this sub-field depends on the Header Extension Type. For fixed-length Header Extensions, the

HEC is 24 bits. For variable-length Header Extensions, the HEC field has variable size, as specified by the HEL field. Note that the length of each Header Extension field MUST be a multiple of 32 bits. Also note that the total size of the Tetrys header, including all Header Extensions and all optional header fields, cannot exceed 255 32-bit words.

4.2. Source Packet Format

A source packet is the encapsulation of a source symbol, a source symbol ID and a Common Packet Header. The source symbols can have variable sizes.

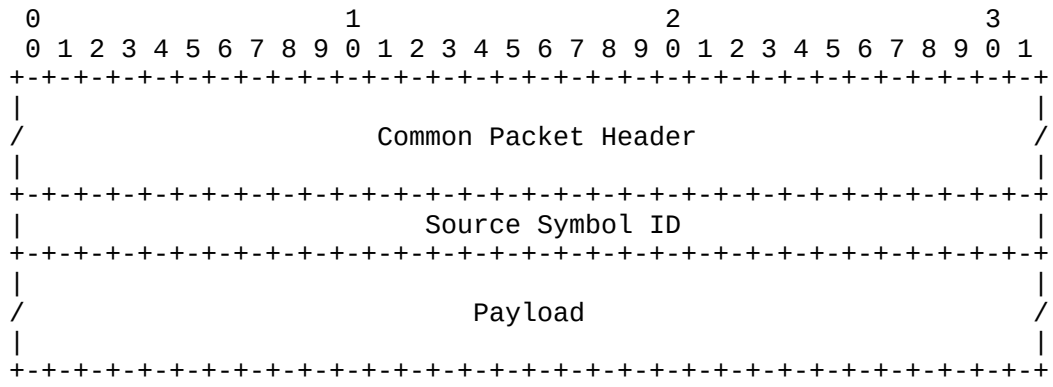


Figure 4: Source Packet Format

Common Packet Header: a common packet header where Packet Type=0.

Source Symbol ID: the sequence number to identify a source symbol.

Payload: the payload (source symbol)

4.3. Coded Packet Format

A coded packet is the encapsulation of a coded symbol, a coded symbol ID, the associated encoding vector and the Common Packet Header. As the source symbols can have variable sizes, each source symbol size need to be encoded, and the result must be stored in the coded packet. The Encoded Payload Size is 16 bits.

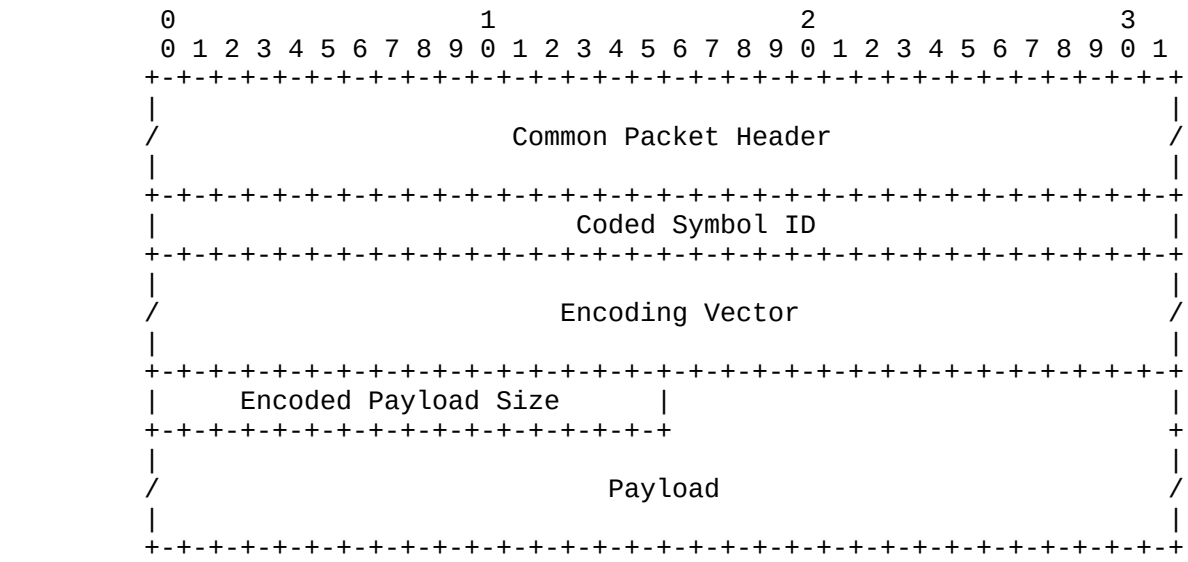


Figure 5: Coded Packet Format

Common Packet Header: a common packet header where Packet Type=1.

Coded Symbol ID: the sequence number to identify a coded symbol.

Encoding Vector: an encoding vector to define the linear combination used (coefficients, and source symbols).

Encoded Payload Size: the coded payload size used if the source symbols are of variable size.

Payload: the coded symbol.

4.4. Acknowledgement Packet Format

A Tetrys Decoding Building Block MAY send back to a Tetrys Encoding Building Block some Acknowledgement packets. They contain information about what it is received and/or decoded, and other information such as a packet loss rate or the size of the decoding buffers. The acknowledgement packets are OPTIONAL hence they could be omitted or lost in transmission without impacting the basic protocol performance.

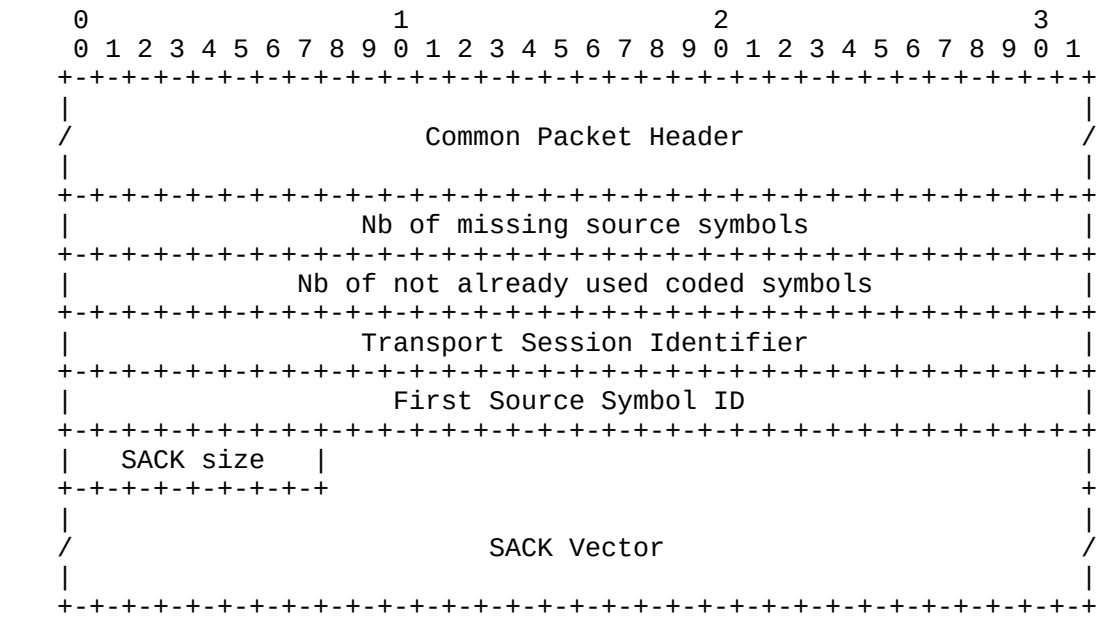


Figure 6: Acknowledgement Packet Format

Common Packet Header: a common packet header where Packet Type=2.

Nb missing source symbols: the number of missing source symbols in the receiver.

Nb of not already used coded symnbols: the number of not already used coded symbols in the receiver that have not already been used for decoding. Meaning the number of linear combinations containing at least 2 unknown source symbols.

Transport Session Identifier (TSI): the unique identifier for the session (CAN be 0bit, depending of the Common Packet Header’s field S)

First Source Symbol ID: ID of the first source symbol to acknowledge.

SACK size: the size of the SACK vector in 32-bit words. For instance, with value 2, the SACK vector is 64 bits long.

SACK vector: bit vector indicating the acknowledged symbols following the first source symbol ID. The "First Source Symbol" is not included in this bit vector. A bit equal to 1 at position i means that the source symbol of ID equal to "First Source Symbol ID" + i + 1 is acknowledged by this acknowledgment packet.

5. The Coding Coefficient Generator Identifiers

5.1. Definition

The Coding Coefficient Generator Identifiers define a function or an algorithm to build the coding coefficients used to generate the coded symbols. They MUST be known by all the Building Blocks.

5.2. Table of Identifiers

0000: GF256 Vandermonde based coefficients. Each coefficient is build as alpha^((source_id*repair_id) % 255).

0001: GF16 Vandermonde based coefficients. Each coefficient is build as alpha^((source_id*repair_id) % 15).

0010: SRLC.

Others: To be discussed.

6. Tetrys Basic Functions

6.1. Encoding

At the beginning of a transmission, a Tetrys Encoding Building Block MUST choose an initial code rate (added redundancy) as it doesn’t know the packet loss rate of the channel. In steady state, the Tetrys Encoding Building Block generates coded symbols when it receives some information from the decoding blocks.

When a Tetrys Encoding Building Block needs to generate a coded symbol, it considers the set of source symbols stored in the Elastic Encoding Window. These source symbols are the set of source symbols which are not yet acknowledged by the receiver.

A Tetrys Encoding Building Block SHOULD set a limit of the Elastic Encoding Window size. This allows to reduce the complexity by considering less source symbols. It also provides a coping mechanism if all the acknowledgment packets are lost.

At the generation of a coded symbol, the Tetrys Encoding Building Block generates an encoding vector containing the IDs of the source symbols stored in the Elastic Encoding Window. For each source symbol, a finite field coefficient is determined using a Coding Coefficient Generator. This generator CAN take as input the source symbol ID and the coded symbol ID and CAN determine a coefficient in a deterministic way. A classical example of such deterministic function is a generator matrix where the rows are indexed by the

source symbol IDs and the columns by the coded symbol IDs. For example, the entries of this matrix can be built from a Vandermonde structure, like Reed-Solomon codes, or from a sparse binary matrix, like Low-Density Generator Matrix codes. Finally, the coded symbol is the sum of the source symbols multiplied by their corresponding coefficients.

6.1.1. Encoding Vector Formats

The encoding vectors are sent in each coded symbols. They contain the source symbol IDs and/or the coefficients.

To avoid the overhead of transmitting all the source symbol IDs, the following algorithm is used to compress them.

6.1.1.1. Transmitting the source symbol IDs

The source symbol IDs are organized as a sorted list of 32-bit integers. Instead of sending the full list, a differential transform to reduce the number of bits needed to represent an ID is used.

6.1.1.1.1. Compressing the Source symbol IDs

Assume the symbol IDs used in the combination are: [1..3],[5..6],[8..10].

1. Keep the first element in the packet as the first_source_id: 1.

2. Apply a differential transform to the others elements ([3,5,6,8,10]) which removes the element i-1 to the element i, starting with the first_source_id as i0, and get the list L => [2,2,1,2,2]

3. Compute b, the number of bits needed to store all the elements, which is ceil(log2(max(L))): here, 2 bits.

4. Write b in the corresponding field, and write all the b * [(2 * NB blocks) - 1] elements in a bit vector, here: 10 10 01 10 10.

6.1.1.1.2. Decompressing the Source symbol IDs

When a Tetrys Decoding Building Block wants to reverse the operations, this algorithm is used:

1. Rebuild the list of the transmitted elements by reading the bit vector and b: [10 10 01 10 10] => [2,2,1,2,2]

2. Apply the reverse transform by adding successively the elements, starting with first_source_id: [1,1+2,(1+2)+2,(1+2+2)+1,...] => [1,3,5,6,8,10]

3. Rebuild the blocks using the list and first_source_id: [1..3],[5..6],[8..10].

6.1.1.2. Encoding Vector Format

The encoding vector CAN be used to store the source symbol IDs included in the associated coded symbol, the coefficients used in the combination, or both. It CAN be used to send only the number of source symbols included in the coded symbol.

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

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

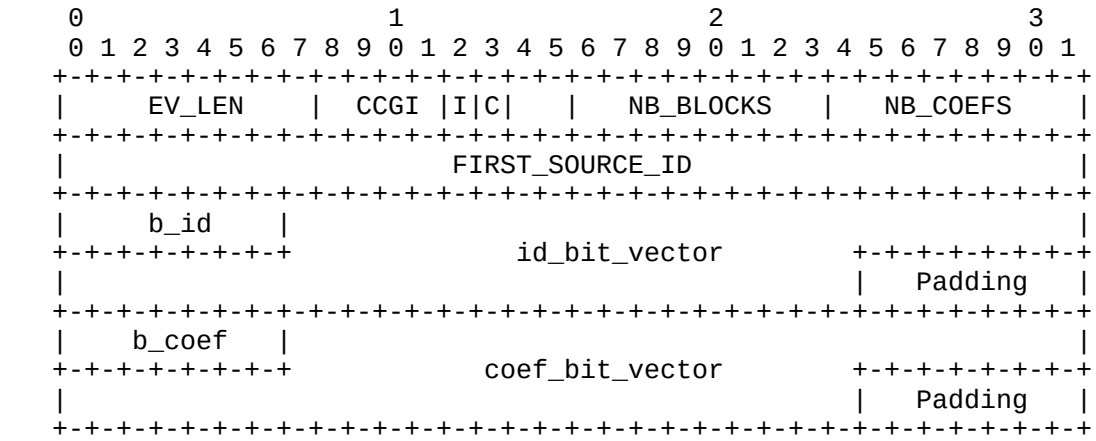


Figure 7: Encoding Vector Format

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

- Coding Coefficient Generator Identifier (CCGI): 8-bit ID to identify the algorithm or the function used to generate the coefficients (see Section 5). As a CCGI is included in each

encoded vector, it can dynamically change between the generation of 2 coded symbols.

- Store the IDs flag (I): 1 bit to know if an encoding vector contains the list of the IDs used. MUST be 1 if the Encoding Vector stores the source symbol IDs.

- Number of blocks used to store the source symbol IDs (NB_BLOCKS): the number of blocks used to store all the source symbol IDs.

- Number of coefficients (NB_COEFS): The number of the coefficients used to generate the associated coded symbol.

- The first source Identifier (FIRST_SOURCE_ID): the first source symbol ID used in the combination.

- Number of bits for each edge block (b_id): the number of bits needed to store the edge (see Section 6.1.1.1).

- The compressed edge blocks (id_bit_vector): equal to b_id * (NB_BLOCKS * 2 - 1).

- Number of bits needed to store each coefficient (b_coef): the number of bits used to store the coefficients.

- The coefficients (coef_bit_vector): The coefficients stored (as a vector of b_coef * NB_COEFS).

- Padding: padding to have an Encoding Vector size multiple of 32-bit (for the id and coefficient part).

6.1.2. The Elastic Encoding Window

When an input source symbol is passed to a Tetrys Encoding Building Block, it is added to the Elastic Encoding Window. This window MUST have a limit set by the encoding building Block (depending of the use case: unicast, multicast, file transfer, real-time transfer, ...). If the Elastic Encoding Window reached its limit, the window slides over the symbols: the first (oldest) symbols are removed. Then, a packet containing this symbol can be sent onto the network. As an element of the coding window, this symbol is included in the next linear combinations created to generate the coded symbols.

As explained below, the receiver sends periodic feedback indicating the received or decoded source symbols. In the case of a unicast transmission, when the sender receives the information that a source symbol was received and/or decoded by the receiver, it removes this symbol from the coding window. In a multicast transmission, a source

symbol is removed from the coding window only when all the receivers have received or decoded it.

6.2. Decoding

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

7. Security Considerations

N/A

8. Privacy Considerations

N/A

9. IANA Considerations

N/A

10. Acknowledgments

N/A

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

11.2. Informative References

[AHL-00] Ahlswede, R., Ning Cai, , Li, S., and R. Yeung, "Network information flow", IEEE Transactions on Information Theory vol.46, no.4, pp.1204,1216, July 2000.

[FECFRAME] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", Request for Comments 6363, October 2011.

[NWCRG-ARCH] NWCRG, , "Network Coding Architecture", TBD TBD.

[RMT] Vicisano, L., Gemmel, J., Rizzo, L., Handley, M., and J. Crowcroft, "Forward Error Correction (FEC) Building Block", Request for Comments 3452, December 2002.

[Tetrys] Lacan, J. and E. Lochin, "Rethinking reliability for long-delay networks", International Workshop on Satellite and Space Communications 2008 (IWSSC08), October 2008.

Authors' Addresses

Jonathan Detchart
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4 31055
France

Email: jonathan.detchart@isae.fr

Emmanuel Lochin
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4 31055
France

Email: emmanuel.lochin@isae.fr

Jerome Lacan
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4 31055
France

Email: jerome.lacan@isae.fr

Vincent Roca
INRIA
655, av. de l'Europe
Inovallee; Montbonnot
ST ISMIER cedex 38334
France

Email: vincent.roca@inria.fr
URI: http://privatics.inrialpes.fr/people/roca/

Network Coding RG
Internet-Draft
Intended status: Informational
Expires: April 3, 2016

B. Khasnabish
ZTE TX, Inc.
S. Sivakumar
Cisco Systems Inc.
E. Haleplidis
University of Patras
C. Adjih
Inria

October 1, 2015

Impact of Virtualization and SDN on Emerging Network Coding
draft-khasnabish-nwcrg-impact-of-vir-and-sdn-04.txt

Abstract

Network Coding is a technique used to code packets and be able to recover coded packets from loses. It requires at least two participating nodes in the path of the packet, one to encode and another to decode. This document discusses the impact of virtualization and Software-Defined Networking (SDN) on the emerging network coding. This document also discusses the integration of network coding in various layers of the network stack and the APIs required from the network coding entity to program it from a controller.

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 http://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 3, 2016.

Table of Contents

1. Introduction

Background:

Abstraction/Virtualization of the Elements of Network:

Control of Network Coding:

APIs:

1.1. Scope

The scope of this document is discussion (and standardization) of utilizing virtualization and SDN paradigm in the emerging network coding.

Ongoing discussions on virtualization and SDN can be found in the following IETF and IRTF Websites: NVO3 [http://datatracker.ietf.org/wg/nvo3/], ForCES [http://datatracker.ietf.org/wg/forces/], I2RS [http://datatracker.ietf.org/wg/i2rs/], SCIM [http://datatracker.ietf.org/wg/scim/], SPRING [http://datatracker.ietf.org/wg/spring/], SFC/NSC [http://datatracker.ietf.org/wg/sfc/], and SDN-RG [http://irtf.org/sdnrg].

Virtualization has been discussed (and deployed) widely in the Computing Industry (e.g., server) in the context of efficient utilization of server resources.

Virtual resources management in the context of Cloud and Data Center (DC) environment using unified API has been discussed in [I-D.junsheng-opsawg-virtual-resource-management].

IETF ForCES Logical Function Block (LFB) Subsidiary Management (SM) for supporting virtualization of ForCES Network Element (NE) including control Element (CE) and Forwarding Element (FE) has been recently discussed in [I-D.khs-forces-lfb-subsidiary-management].

1.2. Abbreviations

- API: Application Programming Interface
- CPSI: Control Plane Southbound Interface
- DAL: Device and resource Abstraction Layer
- DC: Data Center

- NC: Network Coding
- NCC: Network Coding Controller
- NE: Network Element
- PL: Protocol Layer
- SCTP: Stream Control Transmission Protocol
- SDN: Software-Defined Network/Networking
- TCP: Transport Control Protocol
- TML: Transport Mapping Layer
- VCE: Virtual CE
- VDC: Virtual DC
- VNE: Virtual NE

1.3. Conventions and Definitions

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].

The following definitions are taken from the notional Network Coding Architecture slides (http://www.ietf.org/proceedings/88/slides/slides-88-nwcrg-6.pdf). These are repeated here for convenience.

- APP --
- APP Interface --
- Network Coding Transport Protocol --
- Network coding Aware Routing Protocol --
- Link Layer / MAC --
- Others --

2. Separation of Control

There are many advantages of separating control from forwarding, routing, transport, etc. in the emerging SDNs. The ability to integrate network coding in different layers provides the abstraction and the flexibility to choose to apply the technique based on different application characteristics.

In addition to flexibility, this also offers additional reliability and scalability with minimal additional burden on cost and performance.

2.1. Separation Fundamentals

Recent work in the SDNrg have focused on the terminology and a base layered architecture, described in [I-D.irtf-sdnrg-layer-terminology]. [I-D.irtf-sdnrg-layer-terminology] provides a detailed description of the SDN layers architecture by separating SDN into distinct planes, abstraction layers and interfaces.

[I-D.irtf-sdnrg-layer-terminology] describes a number of different planes. The forwarding and operational plane associated with the device, the control and management plane and the application plane. In addition [I-D.irtf-sdnrg-layer-terminology] specifies their relevant interfaces and their characteristics as well as the abstraction layers that all comprise an SDN architecture.

This document is well aligned with [I-D.irtf-sdnrg-layer-terminology]. Depending on where the network coding entity is located, in the forwarding or operational plane or as a service in the control plane different abstraction layers and interfaces are involved.

For example if a network coding entity is located in the forwarding plane of the device, the operations of the network coding entity are described by the Device and resource Abstraction Layer (DAL) and the Network Coding Controller, described in Figure 1 and Figure 2, is a service of the control plane and uses a Control Plane Southbound Interface (CPSI) to control the network coding entity.

2.2. Separation of Control for Transport

In this section we discuss how the separation of control for transport impacts the network coding and its implementation in the emerging software-defined networks or SDNs.

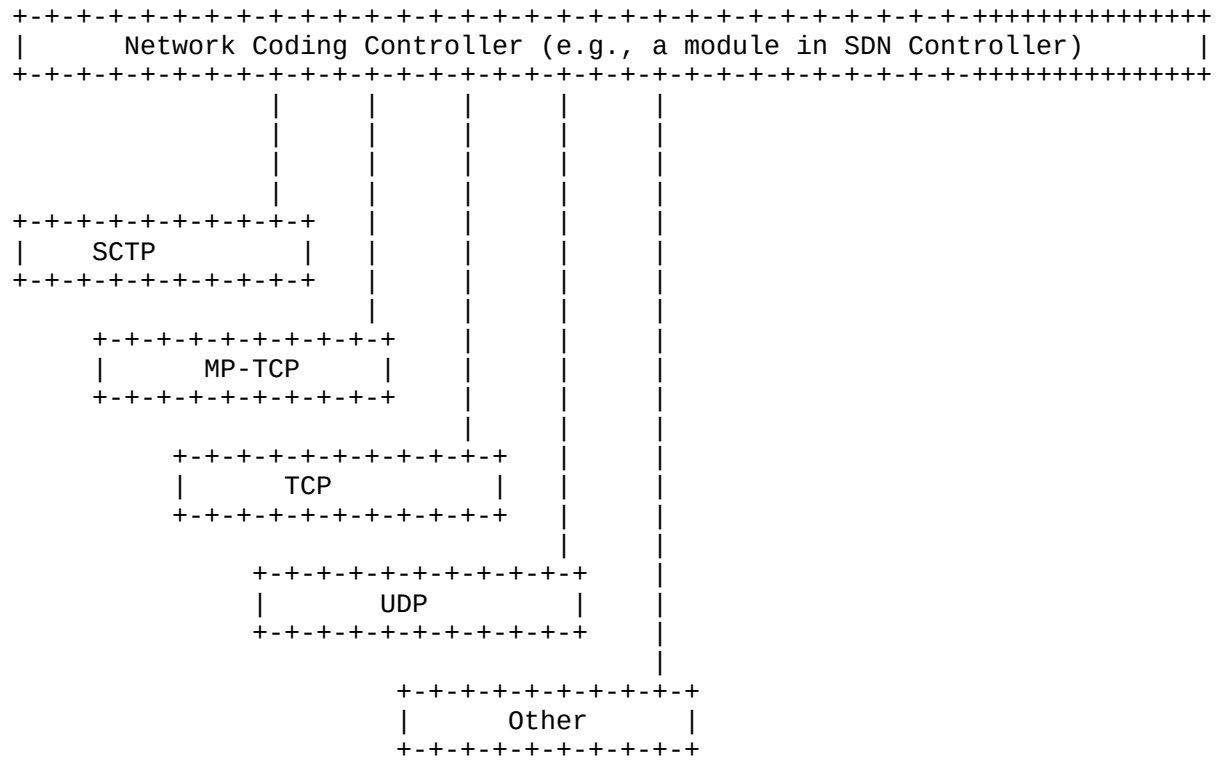


Figure 1

The network coding can be applied for different transport protocols based on what the controller specifies, as shown in Figure 1. For example, the controller can orchestrate the network coding entity to code all the traffic on specific TCP/UDP sockets. This requires the APIs from network coding controller (NCC) to program the network coding function in the forwarding plane to intercept the interesting transport layer packets and code them.

2.3. Separation of Control for network layer

In this section we discuss how the separation of control for routing impacts the network coding and its implementation in the emerging software-defined networks or SDNs.

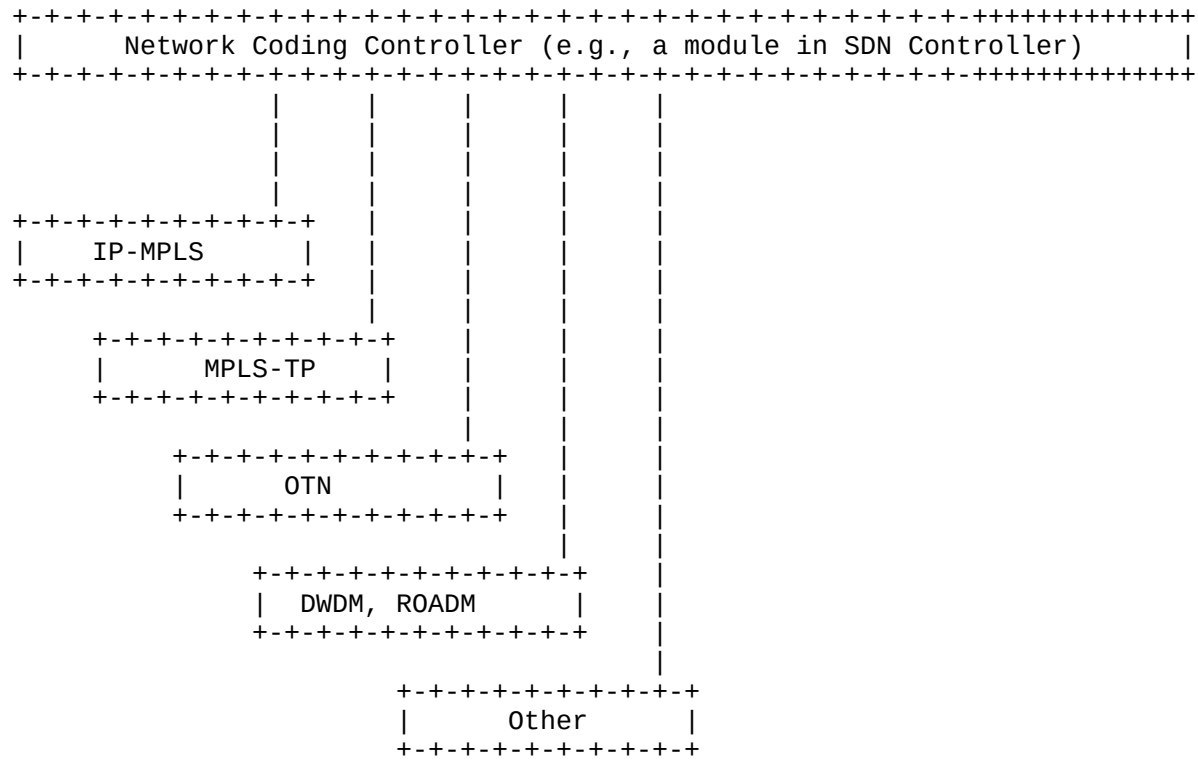


Figure 2

The network coding can be applied on the originating node and/or in intermediate forwarding nodes at the Layer 3 as shown in Figure 2. The NCC can orchestrate the network coding entity running at the network layer to code packets with desired granularity. The granularity can be, for example, all the IP and MPLS packets must be coded. A more granular example, all the MPLS packets matching a certain MPLS label should be coded or a five-tuple in the IP must be matched to determine if the packet is to be coded or not.

2.4. Separation of Control for Forwarding

In this section we discuss how the separation of control for forwarding impacts the network coding and its implementation in the emerging software-defined networks or SDNs.

3. Virtualization and its use in Network Coding

In this section, we discuss general virtualization of applications/ services, and computing/networking resources. We then explore the impact of virtualization on emerging networking coding (architecture, control, and services).

3.1. Virtualization of Application/Service Resources

Virtualization of Application/Service resources is becoming increasingly popular with the proliferation of the APP based services in the mobile and Tablet world.

3.2. Virtualization of Computing Resources

Virtualization of computing resources has been widely used in Cloud Computing [I-D.khasnabish-cloud-reference-framework] environment.

3.3. Virtualization of Network-Level Resources

In this section we discuss virtualization of network resources. The network resources typically include routers, switches, and topology and routing databases, policy and security controllers, etc.

3.4. Virtualization for Network Coding

In this section we discuss virtualization for network coding, its benefits and implementation and management hurdles.

3.5. Network Coding Controller and APIs

In this section we discuss the features/functions of the Network Coding Controller (NCC), and possible NCC APIs. Although North- and South-bound APIs are the most important ones, East, West, etc. bound APIs may be also very useful.

4. Network Coding Control and SDN

In this section we discuss a high-level architecture for network/ service function virtualization and Software-Defined Networking.

4.1. Apps and Service Layer

In this section we discuss the elements and capabilities of the Application and Service layer.

4.2. Control Layer

In this section we discuss the features/functions and the capabilities of the Control layer.

4.3. Virtualization Layer

In this section we discuss the details of the virtualization layer.

4.4. Physical Resources Layer

In this section we discuss the elements of the physical layer.

4.5. Management and Orchestration

In this section we discuss efficient management and Orchestration in virtualized multi-technology and multi-admin-domain environments.

4.6. APIs: For Example Transport APIs

For the emerging Network Coding, defining an appropriate API for dynamically selecting application/service based Transport may be the most suitable option. For example, SCTP [RFC4960] may be more suitable than TCP/Multi-Path-TCP [RFC6824] or UDP [RFC0768] or any other variants for some applications/services.

The added flexibility (due to using an open Transport API) will allow guided navigation of sessions/flows through a variety of network operations systems and physical/virtual infrastructure network/ service elements. This will help achieve unified and seamless user experience irrespective of what the underlying network infrastructure is. Further discussion in this area can be found in [I-D.montpetit-transport-strawman].

4.7. Generic Lifecycle Management

In this section we discuss the generic lifecycle management of virtual entities.

5. Practical Considerations with NC and SDN

In this section, we describe some discussions related to the practical integration of network coding with emerging software-defined networks architectures. We start by observing that, on one hand, thanks to network virtualization, network coding might be done transparently with SDN, which offers the advantage of not having to modify the higher level applications, existing protocols such as TCP/ IP, or the network stack inside guest VMs. On the other hand, this

leaves open the question of which entities in the network will actually do the coding/decoding; in addition, not all of the currently advocated uses of NC are necessarily mapped to cases where SDN is used. Thus some of the major questions are:

- In which scenarios (use cases) could NC be used with SDN?
- How to integrate NC with SDN technologies in practice?

5.1. Some Use Cases for NC in SDN

We present some possible scenarios where NC could bring benefits to SDN. Since SDN might be more related to datacenters or RAN virtualization, they could be slightly different to often described NC use cases. For instance, end-to-end (user) Internet video streaming is a typical application for network coding, but at least on the "last link" to the user, it would not not typically use SDN.

A first concrete scenario of how NC can be conceptually integrated with SDN and virtualization, is the example of IETF NVO3 (Network Virtualization over Layer 3) architecture [RFC7365] (see "Generic Reference Model"). One plausible scenario is a tenant with multiple data centers interconnected through WAN, and with networking applications in virtual machines. In the NVO3 architecture, it is possible to create an overlay over the physical WAN network and then set this overlay as the virtual network for the VMs of the data centers. Network virtualization edges (NVE) are pivotal elements in NVO3; they implement L3 (or L2) virtualization functions. When a NVE receives traffic from a VM (tenant system), it identifies the remote NVE that corresponds to the destination and then the associated overlay, it adds an NVO3 overlay encapsulation header, and it sends the resulting packet on the physical network as native IP packet (encapsulating it). Upon receival by remote NVE, the packet is decapsulated and delivered to the proper destination VM.

In this scenario, the virtualized network relies on actual physical WAN links, and one might imagine several benefits from the use network coding in this context, among the traditional benefits:

- reliability: by splitting the flows on several routes, it is possible to provide diversity for the paths, so that in case of failure of some link, other paths would still be available. The benefits of network coding would includes those of traditional erasure coding; the redundancy could be different than 1+1 (that is: one path+one backup path), typically less costly. This is useful mostly for real-time/low latency communication requirements, in cases where the global convergence time for the

network to recover from link failure (and re-route) is considered too high.

- o performance: in the case of multicast traffic, a first step is of course to have optimized multicast routing on the overlay; a second step would be to optimize the WAN links utilization, which could be done by network coding, as examplified by the classic network coding "butterfly" example. Such scenarios are more likely when information is replicated in more two geographically distinct sites.

A second concrete example would be related to the traditional benefits of network coding for wireless communications. There have been a few proposals for the use of wireless links inside the data centers (on relatively short distances, and with well-defined beams), for instance in [Z2012]. The idea is that (at least in part) gigabit wireless links in the 60 GHz range could be used to interconnect racks of the data center, for instance, top-of-the rack. Because wireless links behave in a more complex way than wired Ethernet/ fiber, complexity would be reflected in their management.

In this scenario, benefits of network coding would include:

- o packet loss recovery: it is natural to use all physical layer and MAC techniques (directional antennas, beamforming, MIMO, error coding ...), but then also natural to use network coding, especially considering multiple hops inside the data center.

- o cross-domain coding: through SDN, there is potential for combining NC in the network itself, with NC in the storage for instance.

- o central network optimization: as in SDN, the network coding controller will have the entire knowledge of the virtualized and physical network topology, (including all type of interfaces, wired/wireless), and could better optimize network use.

It is a very similar architecture to the more general efforts proposed to virtualize Radio Access Networks (e.g. LTE and beyond), although more within the scope of IRTF/IETF. When inter-cell interference is considered in RAN architectures, the outcome would be related to the centralized management of network coding in that scenario (network coding controller), hence would be an inspiration.

Finally, in generic scenarios where multiple path routing is possible (e.g. some context as multi-path TCP/NVO3), an open question is whether the "reliability" case could be extended as follows:

- o latency: as noted before, it is possible to improve reliability with respect to node or link failure with (network coding), at a permanent cost of bandwidth (redundancy), by sending coded packets on multiple paths. Going further, if one application has stringent jitter constraints, one could envision considering late packets as "lost", and still recover packets on time, through network coding/decoding.

A question is whether this trade of bandwidth for some gains in latency is worthwhile for current SDN applications.

5.2. Integrating NC with SDN technologies

Independently of the scenarios and the benefits of NC for SDN, a practical question is which network entity would actually be responsible for performing the (network) coding/decoding. The general issue is that current SDN switches cannot perform network coding at low level, whereas performing it in the controller, could be inefficient.

A first example is taken from a demo at SIGCOMM 2012 by Nemeth et al. [N2012]. Noting that the limited abilities of (OpenFlow) network forwarding devices render difficult the implementation of unconventional techniques, they extend OpenFlow to include new actions in the forwarding engine: precisely, they "extended the OF protocol with three simple actions to support XOR-based mixing of two flows." The global result is not L3 routing, nor L2 switching, but actual network coding done in extended-OpenFlow (non standard). Then a SDN centralized control plane features an explicit "NOX NC controller". Prior work on network coding (without SDN) discussed how to reap bandwidth benefits from creating "butterflies" in the network. For the demo, the NC controller indeed created a butterfly through programming of the (network) coding of video flows from two multicast video streams.

In [L2014], one can find NCoS, a more detailed description of a similar approach, with:

- o A centralized controller has knowledge of the topology and of the flots: it constructs multipath multicast trees (subgraphs), and computes encoding matrix, and then NC flow entries for each switch

- o An extension of OpenFlow with specific buffers (to hold coded packets, managed by the controller), and specific actions: coding initialization, coding, and decoding

- o An implementation done in the simulator Mininet, by extending OpenvSwitch 1.9.0

The second example is related to the previously presented scenario in NVO3 context. In NVO3, the network virtualization edges (NVE) are performing encapsulation/decapsulation of packets. Ignoring interoperability, performance and implementation issues, these edges would be ideally located entities for performing coding, re-coding, and decoding. Because they are well identified, one could imagining chaining the operation of network coding prior or posterior to encapsulation. Compositional SDN architectures would render the integration of network coding more natural. In the same spirit, and more generally, in NFV (Network Functions Virtualization) architectures, a specific network coding "function" could be envisioned, yielding a more natural implementation next to the (purely) forwarding devices.

The third example is related to implementation of network coded and SDN-controlled massive (virtualized) MIMO for providing highly reliable high-capacity access bandwdth for 5G services, as being discussed in [Z2014, Z2015]. ZTE and KT signed strategic Partnership on 5G in order to explore these further (http://www.reuters.com/article/2015/07/16/zte-corporation-idUSnBw156714a+100+BSW20150716). In addition, ZTE is also contributing to 5G projects in Europe including Horizon 2020 in these areas (http://www.businesswire.com/news/home/20150320005199/en/ZTE-Invited-EU-Commissioner-Contribute-Technology-Expertise, http://www.euractiv.com/sections/infosociety/china-eu-5g-and-internet-future-318016).

6. Testbed Platform

Texts and diagram(s) related to Testbeds will be added in this section.

7. Reference Implementation

Texts and diagram(s) related to Reference implementation(s) will be added in this section.

8. Security Considerations

Although the use virtualization and separation of control and transport (and forwarding) open up the possibility of supporting greater flexibility and scalability, these also make the network resources more vulnerable to abuse and spoofing. For example, the security considerations for virtualized resources in DC environment can be found in [I-D.karavettil-vdcs-security-framework].

9. IANA Considerations

This document introduces no additional considerations for IANA.

10. Acknowledgments

The author(s) would like to thank Victor, Brian, Marie-Jose, and many others for their discussions and support.

11. References

11.1. Normative References

[I-D.irtf-sdnrg-layer-terminology]
Haleplidis, E., Pentikousis, K., Denazis, S., Salim, J., Meyer, D., and O. Koufopavlou, "SDN Layers and Architecture Terminology", draft-irtf-sdnrg-layer-terminology-01 (work in progress), September 2014.

[I-D.junsheng-opsawg-virtual-resource-management]
Chu, J., Khasnabish, B., Qing, Y., and Y. Meng, "Virtual Resource Management in Cloud", draft-junsheng-opsawg-virtual-resource-management-00 (work in progress), July 2011.

[I-D.karavettil-vdcs-security-framework]
Karavettil, S., Khasnabish, B., Ning, S., and W. Dong, "Security Framework for Virtualized Data Center Services", draft-karavettil-vdcs-security-framework-05 (work in progress), December 2012.

[I-D.khasnabish-cloud-reference-framework]
Khasnabish, B., Chu, J., Ma, S., Ning, S., Unbehagen, P., Morrow, M., Hasan, M., Demchenko, Y., and M. Yu, "Cloud Reference Framework", draft-khasnabish-cloud-reference-framework-06 (work in progress), January 2014.

[I-D.khs-forces-lfb-subsidiary-management]
Khasnabish, B., Haleplidis, E., and J. Salim, "IETF ForCES Logical Function Block (LFB) Subsidiary Management", draft-khs-forces-lfb-subsidiary-management-01 (work in progress), July 2014.

[I-D.montpetit-transport-strawman]
Montpetit, M., Zhovnirovsky, I., and B. Reuther, "Transport Services Strawman Architecture", draft-montpetit-transport-strawman-01 (work in progress), February 2014.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

11.2. Informative References

[L2014] Sicheng Liu, Bei Hua, , "NCoS: A framework for realizing network coding over software-defined network", IEEE 39th Conference on Local Computer Networks (LCN) 2014, Sep 2014.

[N2012] Felician Nemeth, Adam Stipkovits, Balazs Sonkoly, Andras Gulyas, , "Towards SmartFlow: Case Studies on Enhanced Programmable Forwarding in OpenFlow Switches", SIGCOMM Demo 2012, Aug 2012.

[RFC3654] Khosravi, H., Ed. and T. Anderson, Ed., "Requirements for Separation of IP Control and Forwarding", RFC 3654, DOI 10.17487/RFC3654, November 2003, <http://www.rfc-editor.org/info/rfc3654>.

[RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, DOI 10.17487/RFC3746, April 2004, <http://www.rfc-editor.org/info/rfc3746>.

[RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization", RFC 7365, DOI 10.17487/RFC7365, October 2014, <http://www.rfc-editor.org/info/rfc7365>.

[RFC7642] LI, K., Ed., Hunt, P., Khasnabish, B., Nadalin, A., and Z. Zeltsan, "System for Cross-domain Identity Management: Definitions, Overview, Concepts, and Requirements", RFC 7642, DOI 10.17487/RFC7642, September 2015, <http://www.rfc-editor.org/info/rfc7642>.

[Z2012] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao and Haitao Zheng, , "Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers", SIGCOMM 2012, Aug 2012.

[Z2014] ZTE, Comm-No2-2014., "Special Issue on Software Defined Networking (http://wwwen.zte.com.cn/endata/magazine/ztecommunications/2014/2/)", June 2014.

[Z2015] ZTE, Comm-No1-2015., "Special Issue on 5G Wireless: Technology, Standard and Practice (http://wwwen.zte.com.cn/endata/magazine/ztecommunications/2015/1/)", March 2015.

Authors' Addresses

Bhumip Khasnabish
ZTE TX, Inc.
55 Madison Avenue, Suite 160
Morristown, New Jersey 07960
USA

Phone: +001-781-752-8003
EMail: vumip1@gmail.com, bhumip.khasnabish@ztetx.com
URI: http://tinyurl.com/bhumip/

Senthil Sivakumar
Cisco Systems Inc.
7100-8 Kit Creek Road
Durham, North Carolina 27709
USA

Phone: +001-919-392-5158
EMail: ssenthil@cisco.com

Evangelos Haleplidis
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

EMail: ehalep@ece.upatras.gr

Cedric Adjih
Inria
Saclay - Ile-de-France research centre
France

EMail: Cedric.Adjih@inria.fr

IRTF Network Coding Working Group
Internet-Draft
Intended status: Experimental
Expires: September 10, 2015

M. Montpetit
MIT
V. Roca
INRIA
J. Detchart
ISAE

March 9, 2015

Dynamic Network Coding
draft-montpetit-dynamic-nc-00

Abstract

This document presents a network coding approach that allows coding decisions to be based on the instantaneous conditions at the network nodes. It uses dynamic rates and coefficients to constantly adapt to local conditions and to allow for provider and application differentiation.

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 http://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 10, 2015.

Copyright Notice

Table of Contents

1. Introduction

Network Coding has proven to be an efficient mechanism to provide increased quality of experience for applications depending on Internet Protocols. Current implementations, be them end-to-end or hop-by-hop, depend on a global decision on the type and applicability of the code. However, the heterogeneous nature of IP networks, the differences between transported traffics and the rise of Information Centric Networks (ICN) and multiple in network caches require to define alternatives to current solutions.

Dynamic Network Coding (DNC) intends to use the characteristics of the rising Internet traffic (Internet of Things, streaming, progressive downloads etc.), the use of in-network caching opportunities, customer and policy management and the changing

dynamics on wireless links. These characteristics will be used to adapt the network coding scheme, rate and coefficients to provide adaptive behavior, differentiation and varying quality of experience. In addition, it will also allow to support emerging Internet Architectures such as the ICN that are considering network coding solutions [ICN].

This draft provides the basic elements of such a dynamic code.

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].

3. Definitions, Notations and Abbreviations

3.1. Definitions

This document uses the following definitions, that are mostly inspired from from [RFC5052], [RFC6363].

-- Editor’s note: most of the definitions should be moved to the future NC Architectural document. --

Input Symbol: a unit of data that is provided as an input to the coding process, in a given coding node. It may be a source symbol or an already encoded repair symbol

Output Symbol: a unit of data that is produced as an output of the coding process, in a given coding node

Source Symbol: an original unit of data, before any coding process is applied

Repair Symbol: an Output Symbol that is not a Source Symbol

Code Rate: the ratio between the number of Input Symbols and the number of Output Symbols at given coding node. The Code Rate belongs to a]0; 1] interval. A Code Rate close to 1 indicates that a small number of Output Symbols have been produced during the encoding process

Systematic Code: NC code in which the Source Symbols are part of the Output Symbols

DNC Packet: a packet (e.g., carried as the payload of a UDP datagram) containing a given Output Symbol plus its associated DNC header.

Packet Erasure Channel: a communication path where packets are either dropped (e.g., by a congested router or because the number of transmission errors exceeds the correction capabilities of the physical layer codes) or received. When a packet is received, it is assumed that this packet is not corrupted

To Be Completed

-- Editor's note: Should we consider the possibility of having several symbols per packet? The DNC packet should be updated accordingly in that case. --

3.2. Notations

This document uses the following notations:

CR denotes the Code Rate

To Be Completed

3.3. Abbreviations

This document uses the following abbreviations:

The following definitions are compatible with the FECFRAME framework [RFC6363] and the NC architecture (COMMENT: reference to be added when available).

NC: Network Coding

DNC: Dynamic Network Coding

PRNG: Pseudo-Random Number Generator

TS: Time Stamp

ESI: Encoding Symbol ID

To Be Completed

4. Dynamic Network Coding (DNC) Principles

Network Coding is based on the linear combination of a number of input symbols (at least 1) into a number of output symbols (at least 1), between the ingress and egress of the network. Several such encoding operations can happen in case of in-network re-coding, or there can be a single encoding operation, especially when it is applied end-to-end. In the RLNC [RLNC], Tetrys [I-D.detchart-nwcrg-tetrys], and Dragoncast [I-D.adjih-dragoncast] instances of Network Coding, the linear combination consists in applying a set of coding coefficients to each input symbol of the current encoding window. Here the coding vectors are chosen in a deterministic manner at each encoding node, for instance based on local criteria, or are generated using a PRNG seed chosen locally and carried along with the output symbol.

The DNC proposal extends this process as follows: the set of coding coefficients is determined based on the FEC_Encoding ID, Codepoint, and TS header fields of the various input packets present in the encoding window, plus the local time. The coding decision therefore depends on time, among other pieces of information.

4.1. About the Use of Timestamps in DNC

Each DNC Packet contains timing information: this can be the TS field of the DNC header, or the timestamp field of an NTP header if already present in the packet. This timing information (noted TS hereafter, no matter its origin) is used to determine the coding coefficients in a coding node. When several DNC packets are present in the encoding window, originating from one or several sources, a decision on which TS will be sent downstream in the network must be taken. Options include keeping the oldest TS value, the newest TS value, or generating a local TS value. It is assumed that the granularity of the TS in choosing the coding coefficients would be to the second tin order to react to instantaneous condition.

It is also possible to use the TS in a wider sense, to link to network operations and coding based police management. This includes the determination of the coding window, Code Rate, Galois field, etc.

> -- Editor’s note: This extension is left for future specifications as it requires closer coordination between network management. policy and coding. --

4.2. About the FEC Encoding ID, Codepoint and Coding Decisions

The FEC Encoding ID is used to identify the type of code (i.e., FEC Scheme) to use at each coding node. This is an integer identifier assigned by IANA. The value of the FEC Encoding ID MAY vary over the time, within the same flow, and/or across the same path between several coding nodes. Different coding decisions can be made by different management entities with different operating constraints (for instance content provider versus network operator).

> -- Editor's note: Having the possibility to change the coding decisions can have major practical technical implications that are not considered for the moment. --

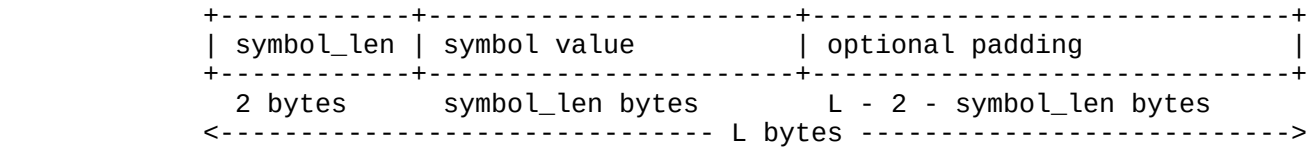
The Codepoint is an opaque value to be used along with the FEC Encoding ID and TS. The {FEC Encoding ID, Codepoint, TS} tuple identifies uniquely the FEC Scheme used and the set of coding coefficients. Examples are provided below on how to do that.

5. Dynamic Network Coding (DNC) Procedures

5.1. Input Symbol Creation

Incoming DNC packets at a coding node are not necessarily of the same fixed size. This size may largely vary over the time, up to a maximum size that is related to the Link MTU and/or Path MTU. This is a problem when Repair Symbols need to be produced by a coding node.

Let us consider the simple case where the FEC Scheme is such that a Repair Symbol necessarily spans the payload of the largest Incoming DNC Packet at the encoding window of the coding node. Let L be equal to the maximum size of the DNC packets in the encoding window plus 2 bytes. The 2 bytes are used to store the original size of the packets. Any DNC packet of size inferior to L-2 bytes MUST be zero padded to achieve the desired length of L bytes. Then any Repair Symbol within the set of Output Symbols is L bytes long. When a Source Symbol is erased and later decoded, the first two bytes of the decoded symbol, that contain the symbol_len field, allows to drop the potential padding.



-- Editor's note: in presence of multiple flows, an additional FlowID field may be prepended in order to identify the flow this Input Symbol belongs to. The details are TBD. --

5.2. Other Procedures TBD

TBD

For instance it may be interesting to have a feedback flow to enable a receiver to adjust its encoding window according to the received or decoded Source Symbols.

6. Application of Dynamic Network Coding to Various Use-cases

-- Editor's note: This section contains material that may be more appropriate in a future NC Architecture document. --

Several technical aspects need to be considered:

- Intra-flow encoding: (single flow) here all the Source Symbols belong to the same and unique flow;
- Inter-flow encoding: (multiple flows) here the Source Symbols belong to two or more flows. This situation is more complex, in particular because it requires to remember the flow a given source symbol belongs to in case this latter gets lost and is recovered by a decoding node;
- End to end: (single coding node) here there is a single coding node and a single decoding node. However the coding and/or decoding nodes are not necessarily the source and destination nodes. For instance these operations can be performed by middle boxes, or coding may be done in a middle box while decoding is performed at the destination node, or vice-versa. The nature of the coding and decoding nodes should not significantly impact the way DNC operates;

  -- Editor's note: This claim is TBC. --

- In network re-coding: (composability) here several coding nodes exist along the path. This situation significantly impacts the way DNC operates, in particular in terms of signaling. Indeed a decoding node MUST be able to identify the exact manner in which a given Repair Symbol has been generated, recursively since this Repair Symbol MAY be the result of several coding operations, at different coding nodes;

- o Multi-source: here several traffic sources exist. They either jointly contribute to the same data flow, all sources sending traffic related to the same content, or they contribute to multiple data flows, sources sending traffic for different contents;

- o Multi-paths: here the traffic follows multiple paths. This traffic can be originated from a unique source (e.g., with multi-path TCP, MPTCP), or with multiple sources which is the more general case;

The above taxonomy can be used to identify several types of use-cases. In the following we consider some of the potential use-cases and explain how DNC can be applied. This is in no case an exhaustive list and will be adapted in the future as we get more insights on the code usage.

6.1. Single Flow, Single Source, End-to-end, Single Path or Multi Paths Use-Case

In this use-case, there is a single flow originated by a single source, with intra stream coding that takes place at a single coding node. There can be either a single path or multiple paths, since this situation does not impact the way DNC operates.

This is the simplest use-case, that is very much inline with currently proposed scenarios for end to end streaming.

6.1.1. Example of DNC Implementation for this Use-Case

The following DNC approach / FEC Scheme is appropriate for this simple use-case. However this is only an example and other approaches may be considered, for instance differing in the way the {FEC Encoding ID, Codepoint, TS} tuple is used.

Let us consider a FEC Scheme that defines the G table containing NL lists, each list being populated with NC coefficients over a given Finite Field, say GF(2^^4). This table, G, is contained in the FEC Scheme specification. Each coding and decoding node supports this FEC Scheme (and potentially a certain number of additional FEC Schemes), this latter being identified by an IANA FEC Encoding ID value.

- o Encoding process: Let W be the encoding window, containing K Input Symbols (constructed as specified in Section 5.1). It is required that K be at most equal to the number of coefficients in each row of G. If this is not the case, an appropriate number of symbols are removed from window W until this property holds. Let TS be

the timestamp corresponding to the current time at the coding node. Let Codepoint be the current integer value of a counter that is managed sequentially, starting a 0 upon initializing the DNC coding node instance, and wrapping to 0 upon reaching the maximum counter value.

-- Editor's note: The counter field size, in bits, is not specified in this version of the document. 32 bits or 16 bits are two possible values. --

Let r be an integer calculated as r=f(Codepoint, TS, K); where f is a deterministic function that produces an integer in the {0; K-1} range. This function is for instance the result of a hash over {Codepoint, TS} modulo K.

-- Editor's note: The FEC Scheme specification fully specifies this f function. --

The output Repair Symbol is computed as the XOR sum of each Input Symbol in W multiplied by the corresponding coefficient in row r of G (i.e., the first symbol is multiplied by G[r][0], the second symbol is multiplied by G[r][1], etc. til the last symbol of W).

- Transmitted Output Symbol: The FEC Encoding ID is communicated in the DNC packet header. The {Codepoint, TS} tuple can be used to uniquely identify the Repair Symbol produced by the coding process. This tuple is communicated in the DNC packet header. The ESI of each packet currently in W is communicated in the DNC packet header. This information can consist of a list of ESIs, or in the simple case where they are all in sequence, the {first ESI, K} tuple can be communicated, or a sequence of such tuples can be sent in the case where ESIs are mostly in sequence with a limited number of gaps, of the first ESI along with a bit field (e.g., of size 64 bits if K is at most 64) can be communicated.

  -- Editor's note: Many other pieces of information will be transmitted for other features. The DNC header format also remains TBD. --

- Processing at the decoding node: Upon receiving this Repair Symbol, an additional equation is added to the current linear system. The FEC Encoding ID enables the decoding node to identify the FEC Scheme used by the coding node, as well as the G matrix. The {Codepoint, TS} tuple enables the decoding node to identify the set of coding coefficients used by the coding node.

- Decoding process: If the linear system enables it, a decoding process is used to recover an erased Source Symbol. The first two

bytes of the decoded symbol is read and used to identify the initial length of the Source Symbol and to remove padding if needed. The decoded Source Symbol is then communicated to the receiving node (or receiving application).

6.2. Single Flow with In-network Re-Coding

In this use-case, there is a single flow, originated either by a single source or by multiple source for the same content, with in-network re-coding capabilities. There can be either a single path or multiple paths (e.g., if there are multiple sources). There are multiple sub use-cases, among which the following three ones.

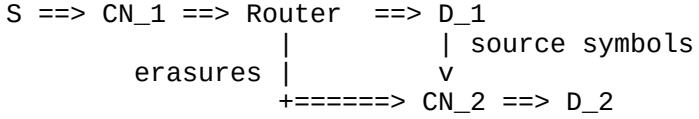


Figure 2

In this first scenario CN_1 et CN_2 are two NC encoders. The flow arriving in CN_2 from the router is impacted by erasures. However this is not the case of the links to destination D_1, that has decoded the packets and this node retransmits the source symbols received or recovered towards D_2. In CN_2 all symbols are recombined and sent to D_2. This could be a scenario that combines wired and wireless paths.

Another scenario is the following, where two sources generate some traffic for the same content:

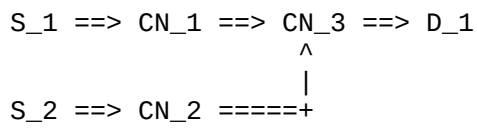


Figure 3

Here S_1 and S_2 are transmitting the same content. The flow coming from S_1 (resp. S_2) is encoded at CN_1 (resp. CN_2), and the two encoded flows are later re-encoded in CN_3, a third NC encoder, on their way to D_1.

Finally, with a single flow passing through wired and wireless paths, the following scenario is likely.

S ==> CN_1 ==> low losses ==> CN_2 ==> high losses ==> D_1

Figure 4

Between CN_1 and CN_2, the network is a wired internet and a high rate code (i.e., adding a limited number of encoded packets) can be used (e.g., it may be a simple XOR of packets as in QUIC [QUIC]). Between CN_2 et D_1 the link can be a WIFI or LTE wireless network, potentially experiencing higher losses. Consequently a higher number of Repair Symbols (i.e., lower code rate) can be needed, with potentially a local feedback loop that enables to adapt the code rate based on the instantaneous conditions observed over that lossy link.

6.3. Other Use Cases

Many use-cases remain TBD, for instance to cover the Peer to peer, complex multipath, or storage use-cases.

7. Acknowledgements

M-J. Montpetit would like to thank Huawei and in particular Shucheng Liu for supporting the work leading to this draft.

8. IANA Considerations

TBD

9. Security Considerations

TBD, see RFC 3552 [RFC3552].

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[I-D.adjih-dragoncast] Adjih, C., Cho, S., and E. Baccelli, "Broadcast With Network Coding: DRAGONCAST", draft-adjih-dragoncast-00 (work in progress), July 2013.

[I-D.detchart-nwcrg-tetrys]
jonathan.detchart@isae.fr, j., Lochin, E., Lacan, J., and V. Roca, "Tetrys, an On-the-Fly Network Coding protocol", draft-detchart-nwcrg-tetrys-00 (work in progress), October 2014.

[ICN] Montpetit, M., Wesphal, C., and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding", Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications (NOM'2012), June 2012.

[QUIC] Roskind, JR., "QUIC: Quick UDP Internet Connections Multiplexed Stream Transport", IETF-88 TSV Area Presentation, November 2013.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, August 2007.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, October 2011.

[RLNC] Ho, T., Koetter, R., Medard, M., Karger, D., Effros, M., Shi, J., and B. Leung, "A Random Linear Network Coding Approach to Multicast", IEEE Transactions on Information Theory Vol. 52 No. 10, October 2006.

Appendix A. Appendix: IPR aspects

IPR aspects if any to be provided later

Authors' Addresses

Marie-Jose Montpetit
MIT
Cambridge, MA 02138
USA

Email: mariejo@mit.edu

Vincent Roca
INRIA
655, av. de l'Europe
Inovallee; Montbonnot
ST ISMIER cedex 38334
France

Email: vincent.roca@inria.fr
URI: http://privatics.inrialpes.fr/people/roca/

Jonathan Detchart
ISAE
10, avenue Edouard-Belin
BP 54032
Toulouse CEDEX 4 31055
France

Email: jonathan.detchart@isae.fr

tsvwg
Internet-Draft
Intended status: Informational
Expires: August 12, 2015

J. Wang
L. Deng
China Mobile

February 8, 2015

Combining TCP with coding in wireless network
draft-wang-tsvwg-tcp-coding-00.txt

Abstract

This drat discusses combining TCP with coding in wireless network. A lot of factors lead to unstable air-link in wireless network, which causes high bit error rate and thus high packet loss. Since packet loss will degrade TCP throughput and coding can erase packet loss in transmission, combing TCP with coding can achieve better performance.

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 http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on August 12, 2015.

Table of Contents

1. Introduction

The TCP was primarily designed for the wired network. In wired networks random BER (bit error rate) is negligible and congestion is the main cause of packet loss. TCP reacts to any packet losses by dropping its transmission (congestion) window size before retransmitting packets, initiating congestion control or avoidance mechanism (e.g., slow start). These measures result in a reduction in the load on the intermediate links, thereby controlling the congestion in the network.

In wireless network, there are lot of factors (e.g., weather conditions, urban obstacles, multi-path interferences, limited coverage, mobility of the handset, etc.,) leading to unstable air-link. As a result, wireless links exhibit much higher BERs than wired links. Since all packet losses are considered as network congestion in standard TCP, packet loss caused by the high BER of the wireless link would trigger the TCP sender to reduce its sending rate unnecessarily. This leads to the drastic decrease of TCP's throughput in the wireless network.

Coding has emerged as an important potential approach to the operation of wireless networks. The major benefit of coding stems from its ability to mix data, across time and across flows. This makes data transmission over lossy wireless networks robust and effective. By applying coding in TCP, the data are coded before transmission and redundancy is introduced during the coding. As a result, the lost packets in the lossy wireless link can be recovered at the receiver side by using redundancy. A famous coding TCP prototype has been introduced by MIT, which combines TCP with network coding and the experiments have shown that the network coding TCP can improve throughput in wireless network significantly [Sundararajan011].

This document introduces a coded TCP mechanism by introducing coding in the TCP layer. The coded TCP can improve data transmission in lossy wireless network. Besides, it can interwork with standard TCP and thus allows incremental deployment.

2. System architecture

The coded TCP adds the data encoding and decoding to the standard TCP. The operation can be divided into sender side and receiver side, and a coding header is introduced.

2.1. Sender side algorithm

- When data arrives from the application, first divide it into packets where each packet is assumed to be of fixed length. The length of packet is set as MSS (Max Segment Size) minus the length of coding header. If the remainder of the data is not large enough to form a complete packet, the packet is padded with zeros to ensure that all packets are of the same length. Then deliver the packets to encoding buffer. The encoding buffer is used to encode packets.
- The encoding buffer consists of coding blocks where each coding block contains the same number of packets. If the remainder of the packets is not enough to fulfill a coding block, the coding block is padded with zeros to ensure that all coding blocks contains the same number of packets.
- In each coding block, the original packets are encoded by using coding algorithm. Many coding algorithms can be applied and RLNC (Random Linear Network Coding) is one of the proved algorithms [Sundararajan011] [Kim013]. The algorithm generates the combinations of the original packets (e.g., RLNC generates the random linear combinations of the original packets). The number of the generated combinations can be greater than the number of the original packets and thus the redundancy is introduced. Each coding block generates the same number of the combinations. The coding algorithm MUST ensure that if the number of the received combinations is equal or exceed to the number of the original packets in a coding block at the receiver side, all original packets in this coding block can be decoded successfully.
- The generated combinations are delivered to the standard TCP stack in order to send to receiver side. In this case, the receiver side receives the encoded packets (combinations) instead of the original packets.

2.2. Coding header

In coded TCP, a coding header is added to each generated combination. The coding header contains the information for the receiver side to decode. Both coding header and the combination are assumed to be the TCP payload. The payload is delivered to the standard TCP stack to send to the receiver side.

The coding header is shown as follows:

[block seqno, orig number, combination number, coding coefficients, block padding, packet padding]

- o block seqno: indicates the sequence number of the coding block, where the combination is generated.
- o orig number: the number of original packets in the coding block.
- o combination number: the number of generated combinations in the coding block.
- o coding coefficients: the coefficients of original packets involved in the combination. The coding coefficients are generated by the coding algorithm and are used by the receiver side to decode.
- o block padding: the number of the padding packets in the coding block.
- o packet padding: the number of the padding bytes in the last non-padding packet in the coding block.

2.3. Receiver side algorithm

- o When packet arrives from the IP stack, get the coding header from the packet. The receiver side records the number of received combinations in each coding block. If the number of received combinations is less than the number of the original packets which is indicated in the coding header, the receiver side invokes the standard TCP stack to reply ack. In this case, the original packets of the coding block cannot be decoded and receiver side needs to receive more combinations; if the number of received combinations is equal to the number of the original packets, the receiver side acknowledges that all combinations generated in the coding block are received. In this case, the original packets of the coding block can be decoded and no need to receive more combinations.
- o The received packet is kept in the decoding buffer. In each coding block, if the number of the received combinations is equal to the number of the original packets, the received combinations are calculated by using coding/decoding algorithm. The original packets can be decoded and sent to the application.
- o The key point of the receiver side algorithm is that in each coding block it can decode the original packets and acknowledges

receiving all combinations if the number of received combinations is equal to the number of original packets. In this case, the coding can erase the packet loss in the wireless link and thus improve TCP throughput in the wireless network.

3. Interworking with standard TCP

The coded TCP transmits the combinations and cannot interwork with standard TCP which transmits original packets. The coded TCP SHOULD be able to choice whether or not enabling coding dynamically. If coding is disabled, coded TCP behaves like the standard TCP. This section proposes a mechanism for coded TCP to interwork with standard TCP. One bit in the reserved field of the TCP header is defined to identify whether transport uses the coded TCP or standard TCP. Setting the bit (called as coding bit) as "1" means that the transport uses coded TCP, otherwise the transport uses standard TCP. The sender and the receiver negotiate at the TCP three-way handshake stage when TCP connection is setting up. The steps are illustrated as follows:

- Both sender and receiver use coded TCP: the sender sets the coding bit in the TCP sync message to identify that the sender supports coded TCP. The receiver sets the coding bit in the TCP sync ack message to identify that the receiver supports coded TCP. In this case, both sender and receiver know that the peer supports coded TCP. Thus, the sender and receiver communicate using coded TCP.
- The sender uses coded TCP and the receiver uses standard TCP: the sender sets the coding bit in the TCP sync message to identify that the sender supports coded TCP. The receiver is the standard TCP and replies the TCP sync ack without setting coding bit. In this case, the sender knows that the peer is the standard TCP and thus the sender and receiver communicate using standard TCP.
- The sender uses standard TCP and the receiver uses coded TCP: the sender is the standard TCP and thus sends the TCP sync message without setting the coding bit. The receiver knows that the sender is the standard TCP and thus communicates with sender with standard TCP.

4. Security Considerations

TBA

5. IANA Considerations

None.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[Kim013] Kim, M., Cloud, J., ParandehGheibi, A., Urbina, L., Fouli, K., leith, D., and M. Medard, "Network Coded TCP (CTCP)", 2013.

[Sundararajan011] Sundararajan, J., Shah, D., Medard, M., Jakubczak, S., Mitzenmacher, M., and J. Barros, "Network Coding Meets TCP: Theory and Implementation", March 2011.

Authors' Addresses

Jinzhu Wang
China Mobile

Email: wangjinzhu@chinamobile.com

Lingli Deng
China Mobile

Email: denglingli@chinamobile.com