Random Linear Network Coding on Programmable Switches

D. Gonçalves¹, S. Signorello¹, F. M. V. Ramos¹, M. Médard²

¹ Faculdade de Cienças, University of Lisbon, Portugal.
² Massachusetts Institute of Technology. (MIT), USA.

EuroP4 2nd European workshop on P4 at ANCS’19, 23th Sep Cambridge, UK.
A Primer on Network Coding & Motivation
Network Coding with an example

Instead of simply forwarding data, nodes may **recombine** several input packets into one or several output packets.

**Benefits over different scenarios:**

Throughput, Robustness, Security.
How far research on NC goes?

Theoretical research on Network Coding

"Network Information Flow" ~ 10K citations
Deployed NC-based systems?

**Software** running in end-hosts: e.g. the Kodo C++ Library

**Overlay** systems: e.g. the Avalanche P2P system (Microsoft)

Software and Overlay, but not in the network data-plane, why?
- Payload processing,
- Complex arithmetic.
Linear Network Coding

Data $P_i$ interpreted as numbers over some finite field $GF(2^s)$

Downside: pre-defined Centralized computation of coefficients.
Random Linear Network Coding

Coefficients randomly chosen in $\text{GF}(2^s)$!

Coefficients (packet header) + coded symbols in output packet
Practical RLNC

Decoding means:

\[
\begin{pmatrix}
P_1 \\
P_2 \\
P_3 \\
P_4 \\
P_5 \\
P_6 \\
\end{pmatrix}
= \begin{pmatrix}
C_1 \\
C_2 \\
C_3 \\
C_4 \\
C_5 \\
C_6 \\
\end{pmatrix}
\]

To reduce complexity, data are divided in smaller blocks over which coding/decoding is performed.

a.k.a. generation-based RLNC
Change in Networks’ Status Quo

Past: *Fixed-Function Switching Chips*

- TCP
- RTP
- IPv4/6
- VLAN
- IETF standards

Future: Programmable Switching Chips

Custom Protocol

**DIY data plane**
"This work proposes a random linear network coding data plane written in P4, as first step towards a production level platform for network coding."

Goal: Understanding the trade-offs for running RLNC functions in the data-plane of the latest programmable switching chips.
Architecture of our Network Coding Switch
RLNC target data plane behavior(s)

1° behavior - coding generations

**Sender**
Sends uncoded data split in generations

**Switch**
Buffers entire generation, creates and forwards linear combinations of symbols

**Receiver**
Acks a generation when that is successfully decoded

2° behavior - recoding generations

**Sender**
Sends coded data split in generations & Related coefficients

**Switch**
Buffers entire generation & coefficients, creates and forwards linear combinations of symbols and recoded coefficients

**Receiver**
Acks a generation when that is successfully decoded
Practical generation-based RLNC

Requirements

Packet Format
To encode symbols/coefficients
And coding parameters

Finite Field (GF) arithmetic
To compute linear combinations
Of the symbols

Buffering
To store all the symbols of a
generation before coding/recoding.
Packet format

Coding parameters header

Symbol representation draft at:

Rcv-based Ack mechanism for generations

Coefficients and symbols
Extracted as P4 packet headers
Buffering

An entire generation must be received and stored before coding can be performed.

State (symbols and coefficients) across packets which must be dynamically indexed by generation id in packet headers.

Where a generation starts (head) and where is the next empty slot (offset).

All implemented with P4 externs (registers).
Galois Field Arithmetic

Random selection of coefficients $c_i$ in GF

$$Y_1 = c_1 \times X_1 + c_2 \times X_1^2 + c_3 \times X_1^3$$

Addition in GF
Equals simple bit-xor

Multiplication in GF
Reducing, through mod, the product of two elements by an irreducible polynomial

Alg1 Compute Intensive
Shift and add operations performed bit-by-bit

Alg2 Memory Intensive
$$\text{mul}(a,b) = \text{antilog}((\log(a) + \log(b)) \text{mod} Q)$$

3 table look-ups, 1 add, 1 mod

Y ~ output symbol
X ~ input symbols
c ~ coefficients
RLNC.p4 on the Target Architecture

Symbols and coefficients buffering

1° Symbols and coefficients extraction

2° GF arithmetic on symbols & coefficients

1° Emitting Coded Symbols & related Coefficients.

2° Linear combinations of the same generation are carried over multiple packets through the target Packet Replication Engine (e.g., using multicast primitives)
Lessons & Evaluation
Set-up for preliminary evaluation

**P4-target**: bmv2’s simple-switch

**Application**: python library for network coding and Scapy for custom pkt header

**Finite Field**: GF(2^8) with variable generation size, # packet symbols, # lin comb

**Correctness**: for every experiment, we check decoding at the receiver side is correct!

**Objective** to gain some preliminary insights about:

- Impact of coding parameters on the P4 program,
- Performance of the tested target with regard to generation size and recoding.
Coding parameters (generation size, field size, # symbols in coded packets...) and GF multiplication algorithm affect code size.

**Solution**: code-generating template

Output:

Alg2 (lookup tables) produces less verbose code & more compact binaries.

\[ \text{mul}(a,b) = \text{antilog}((\log(a) + \log(b)) \mod Q) \]

+ is less resource-intensive (%CPU) on the test target.
RLNC Switch Performance

Take-away: performance drop due to bigger gen sizes and recoding can be addressed
Conclusion and Future Work
Optimizations & Targets & Apps

P4 code and testing suite available soon at: https://github.com/netx-ulx/NC

1. Sparse coding to reduce:
   - packet overhead
   - # of operations

2. Exploring architectural/
   Language support for
   this data-plane behaviors

3. Measuring
   Packet overhead
   Latency
   Network throughput
   in
   Network settings
   With
   Real applications
Thank you! Questions?