Random Linear Network Coding on Programmable Switches

D. Gonçalves[1], S. Signorello[1], F. M. V. Ramos[1], M. Médard[2]

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

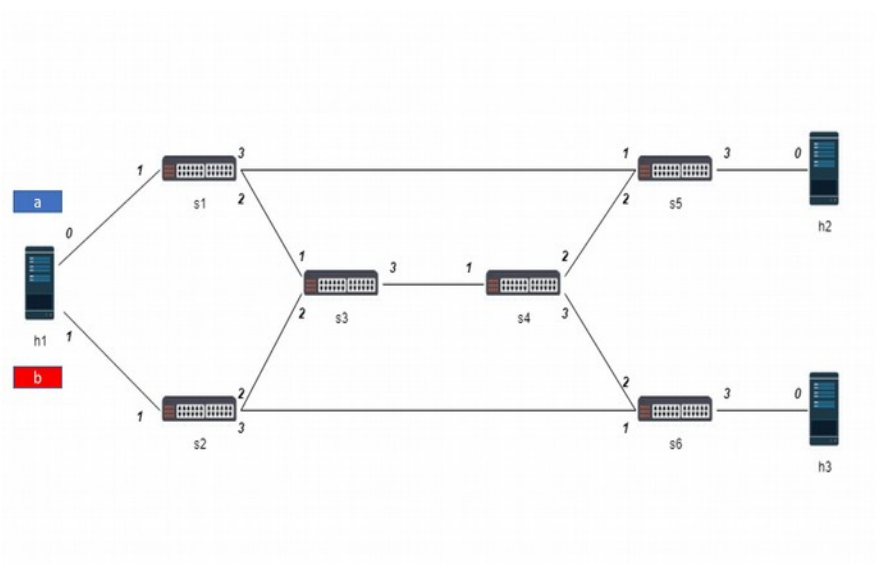
EuroP4 2nd European workshop on P4 at ANCS'19, 23th Sep Cambridge, UK.

A Primer on Network Coding & Motivation

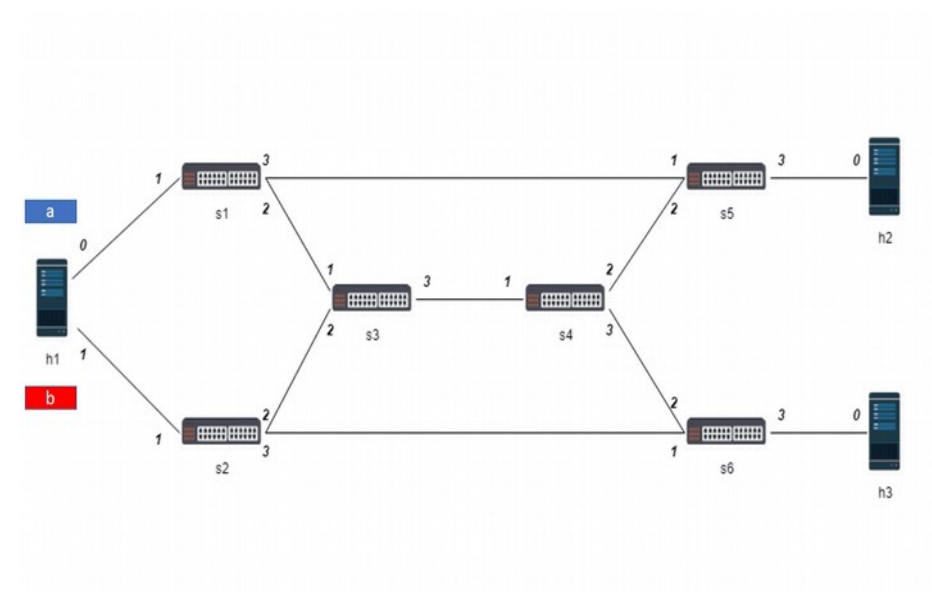
Network Coding with an example

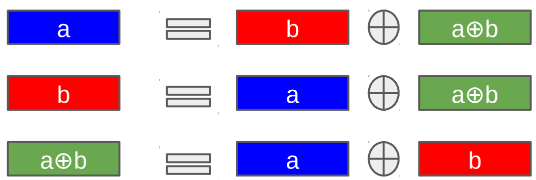
Instead of simply forwarding data, nodes may recombine several input packets into one or several output packets.

Traditional routing solution



Network coding

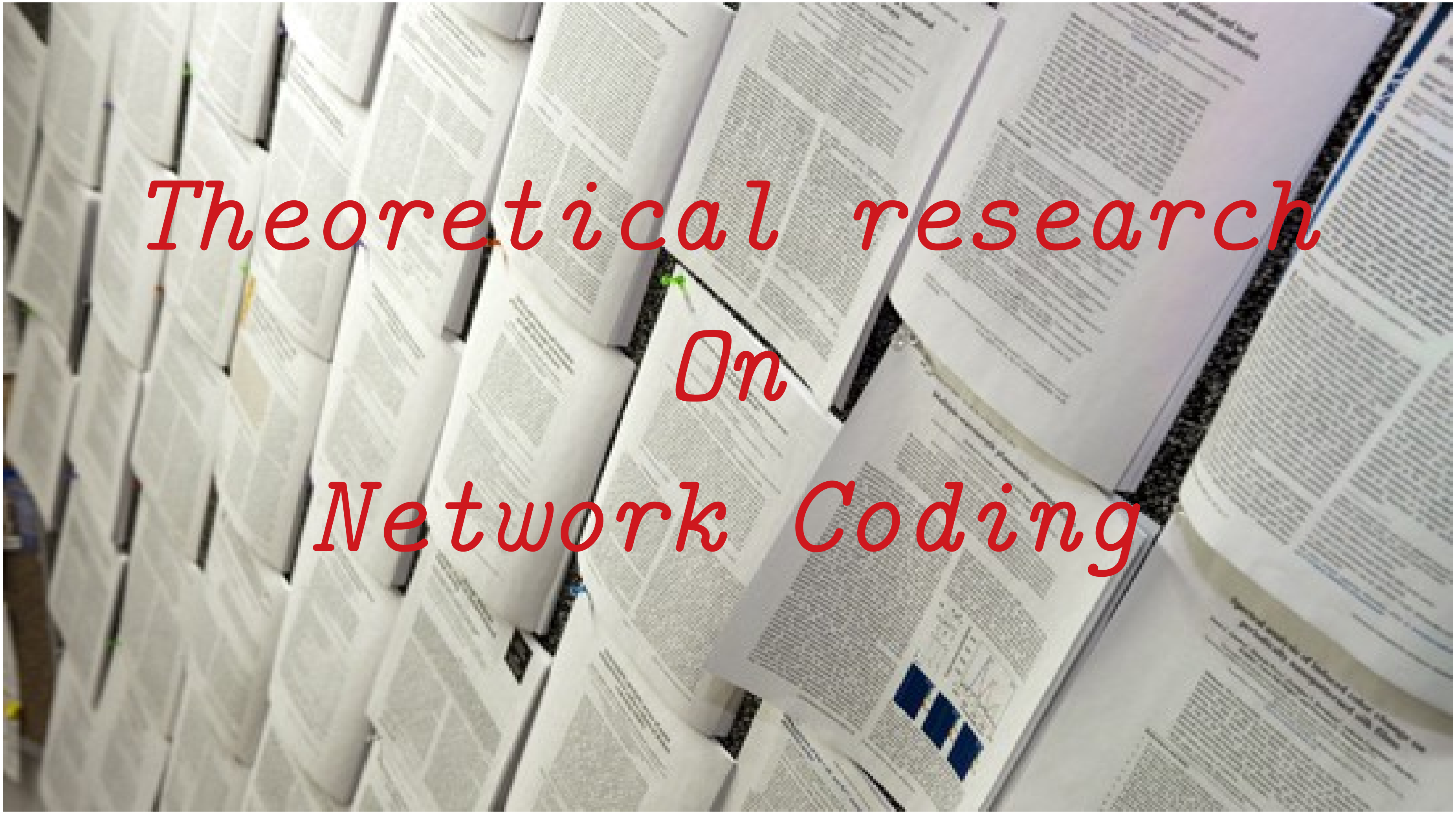




Benefits over different scenarios:

Throughput, Robustness, Security.

How far research on NC goes?



"Network Information Flow" ~ 10K citations

Deployed NC-based systems?

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

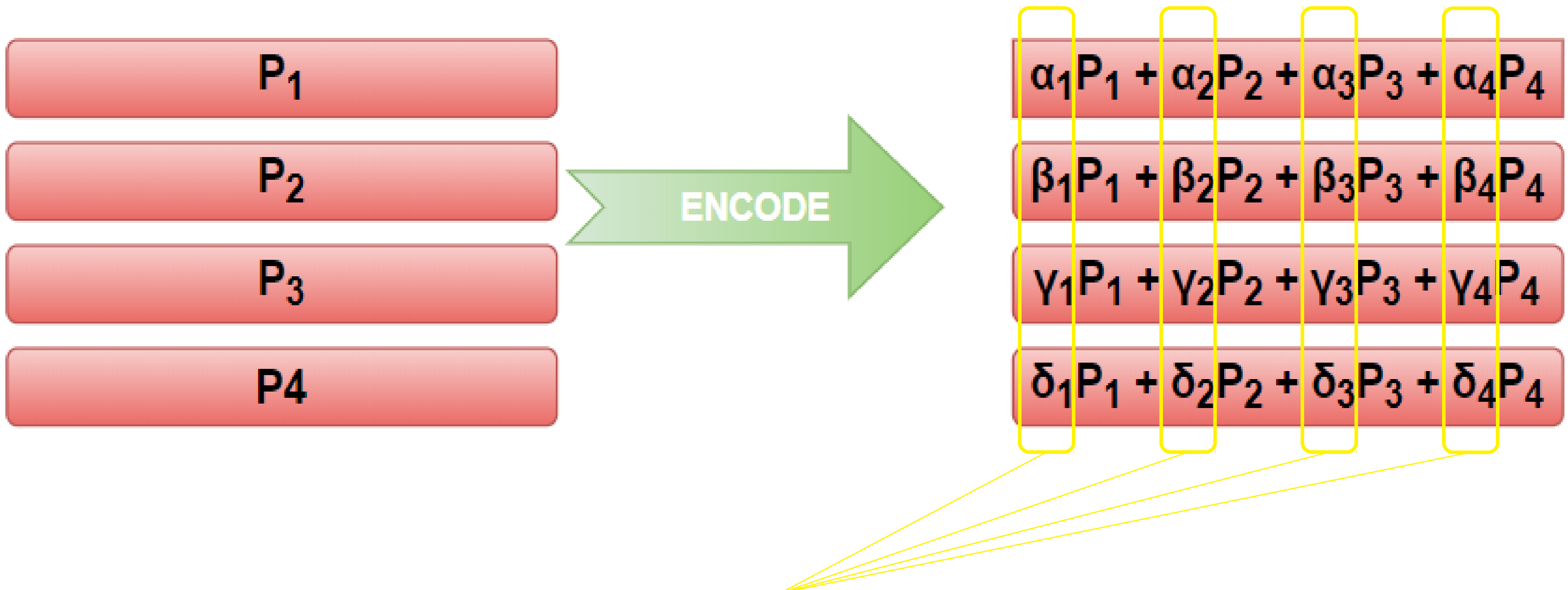
Overlay systems: e.g. the Avalanche P2P system (Microsoft)

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 $GF(2^s)$

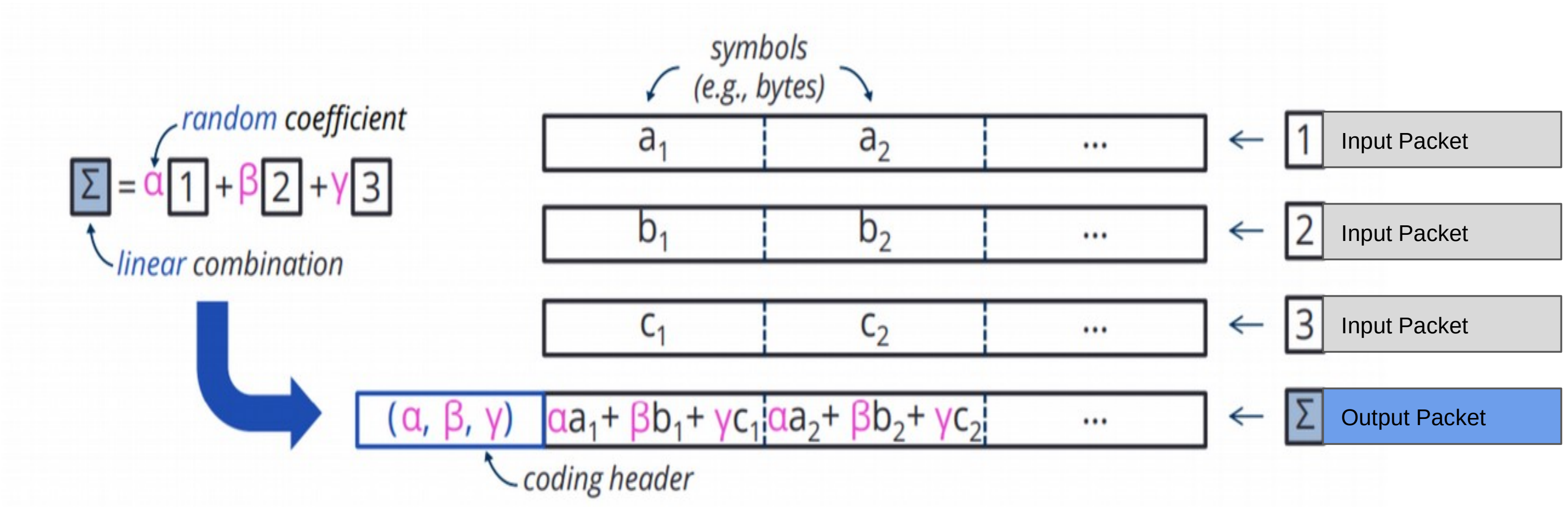


Coefficients **carefully** chosen in $GF(2^s)$!

Downside: pre-defined **Centralized** computation of coefficients.

Random Linear Network Coding

Coefficients **randomly** chosen in $GF(2^s)$!

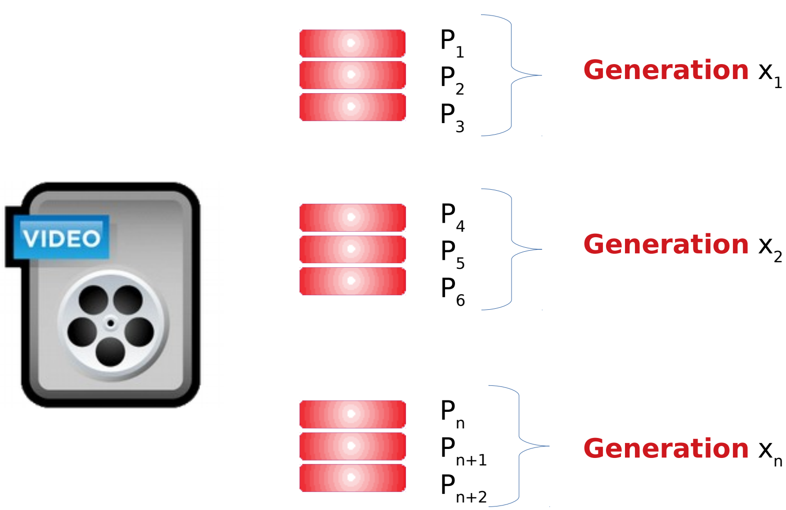


Coefficients (packet header) + coded symbols in output packet

Practical RLNC

Decoding means:

$$\overset{\text{Orignal packets}}{\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{pmatrix}} = \overset{\text{coded packets}}{\begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{pmatrix}} \overset{\text{coding coefficients}}{\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} & \alpha_{1,5} & \alpha_{1,6} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} & \alpha_{2,5} & \alpha_{2,6} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} & \alpha_{3,5} & \alpha_{3,6} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} & \alpha_{4,5} & \alpha_{4,6} \\ \alpha_{5,1} & \alpha_{5,2} & \alpha_{5,3} & \alpha_{5,4} & \alpha_{5,5} & \alpha_{5,6} \\ \alpha_{6,1} & \alpha_{6,2} & \alpha_{6,3} & \alpha_{6,4} & \alpha_{6,5} & \alpha_{6,6} \end{pmatrix}}^{-1}$$

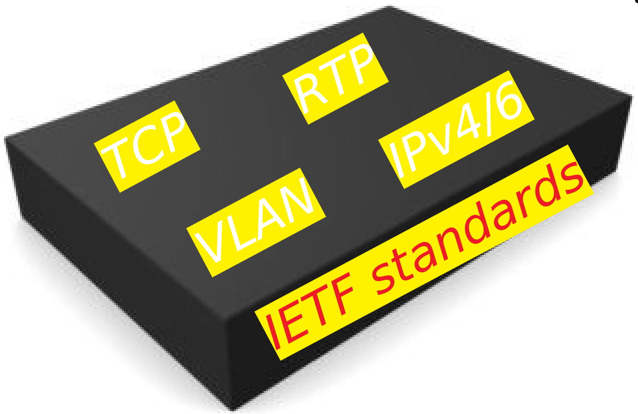


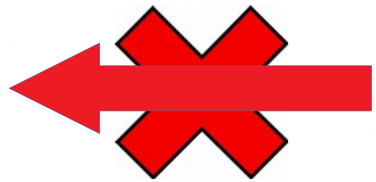
To reduce complexity,
data are divided in **smaller blocks**
over which coding/decoding is perfomed.

a.k.a. **generation-based** RLNC

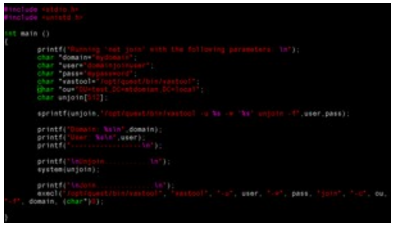
Change in Networks' Status Quo

Past: *Fixed-Function Switching Chips*





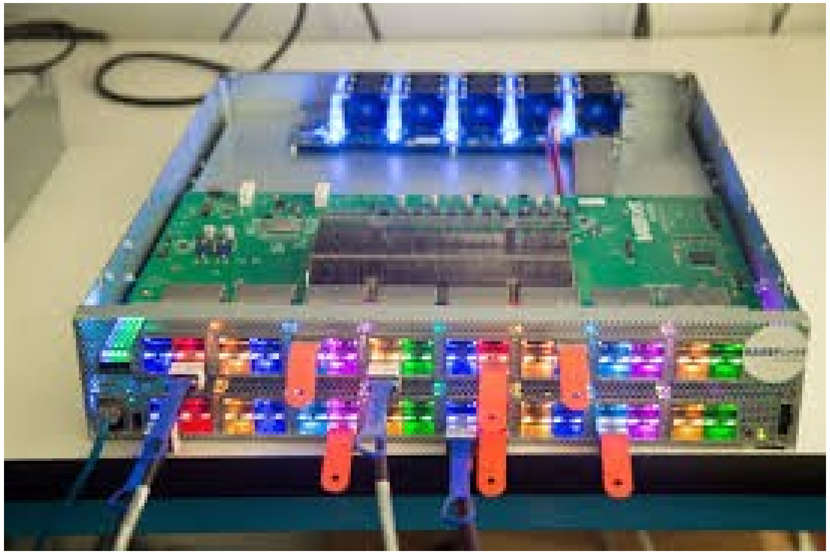
Custom Protocol



Future: Programmable Switching Chips

Practical RLNC in production

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



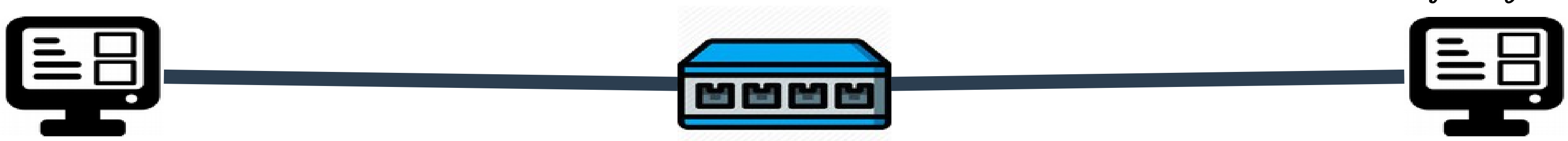
Goal: Understanding the trade-offs for running RLNC functions in the data-plane of the latest programmable switching chips.

Architecture of our Network Coding Switch

RLNC target data plane behavior(s)

1° behavior - coding generations

Receiver
Acks a generation when that is successfully decoded

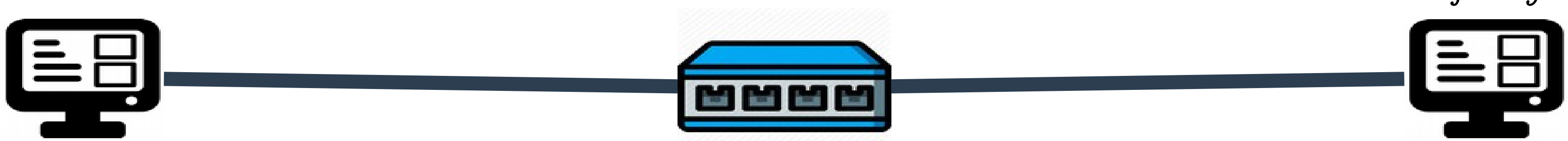


Sender
Sends uncoded data split in generations

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

2° behavior - recoding generations

Receiver
Acks a generation when that is successfully decoded



Sender
Sends coded data split in generations & Related coefficients

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

Practical generation-based RLNC

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

Symbol representation draft at:

https://datatracker.ietf.org/doc/draft-heide-nwcrg-rlnc/

Coding parameters header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| T |SYMBOLS|   ENCODER RANK     |   SEED or CODING COEFFICIENTS |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                       SYMBOL(S) DATA                          |
|                            ...                                |
|                                                               |
+---------------------------------------------------------------+
```

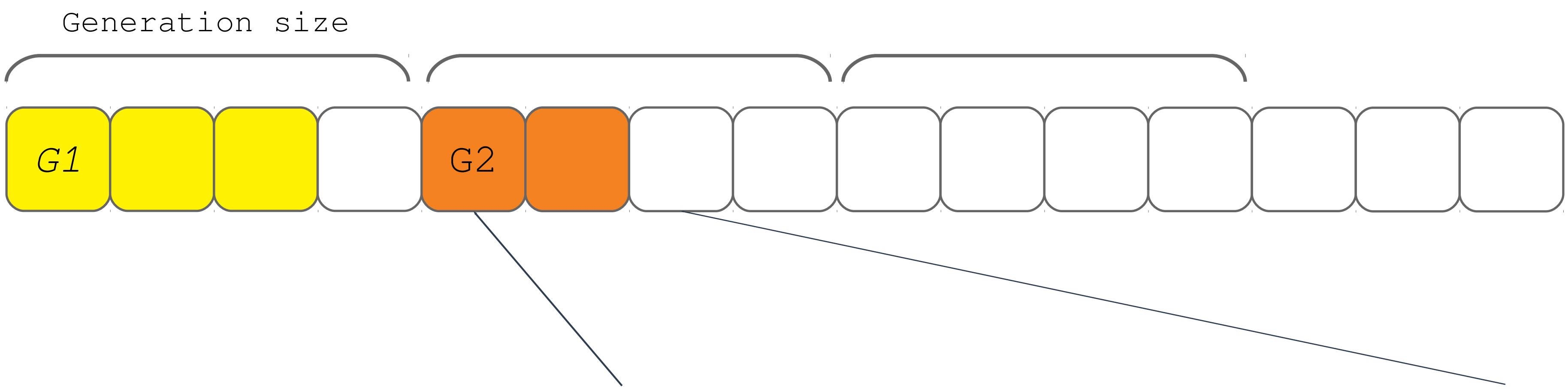
Rcv-based Ack mechanism for generations

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

$$Y_1^1 = c_1 * X_1^1 + c_2 * X_1^2 + c_3 * X_1^3$$

Y ~ output symbol
X ~ input symbols
c ~ coefficients

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

$$mul(a,b) = antilog((\log(a) + \log(b)) \bmod Q)$$

3 table look-ups, 1 add, ~~1 mod~~

RLNC.p4 on the Target Architecture

1° 2°

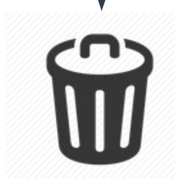
Symbols and coefficients extraction

GF arithmetic on

symbols & coefficients

1° 2°

Symbols and coefficients buffering



Emitting Coded Symbols & related Coefficients.

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.

On Code Size & GF arithmetic

Coding parameters(generation size, field size, # symbols in coded packets…) and GF multiplication algorithm affect code size.

Solution: code-generating template

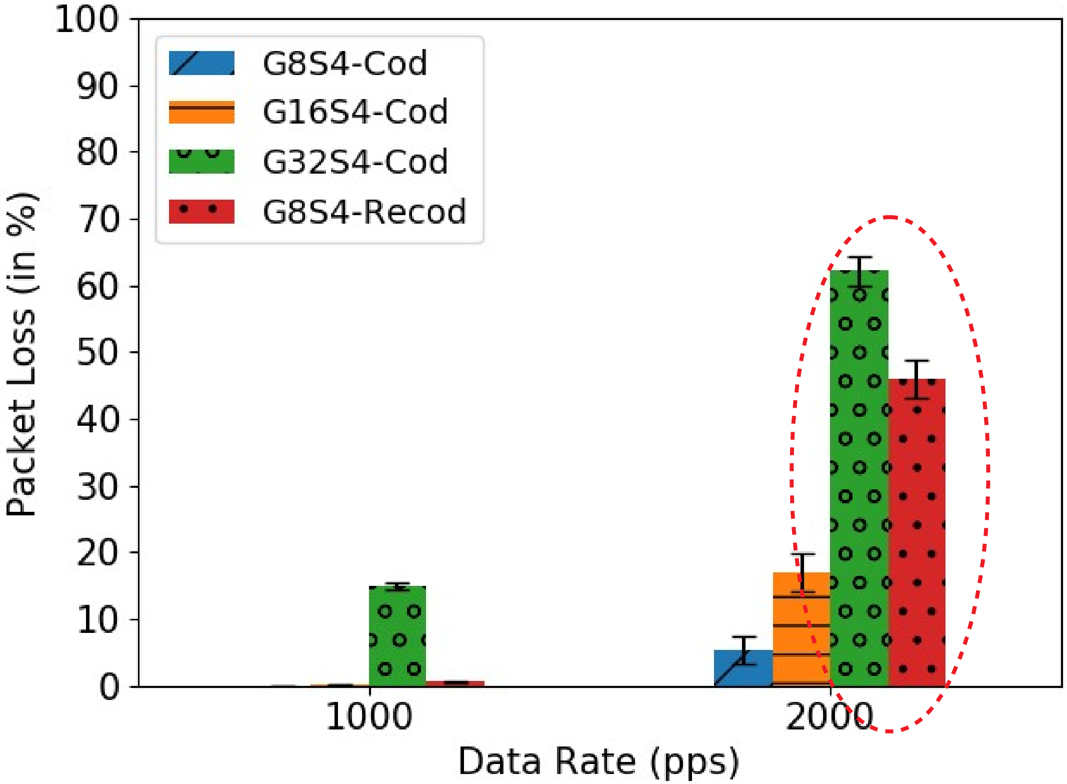
Output:

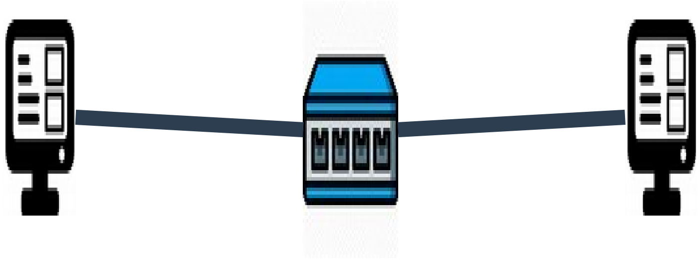
Alg2 (lookup tables) produces less verbose code & more compact binaries.

$$mul(a,b) = antilog((\log(a)+\log(b)) \bmod Q)$$

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

RLNC Switch Performance





Increasing generation size & Recoding => :

- ++buffering
- ++GF arithmetic

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*

Sparse coding to reduce:
packet overhead
&
# of operations

Exploring architectural/
Language support for
this data-plane behaviors

Measuring
Packet overhead
Latency
Network throughput
in
Network settings
With
Real applications

Thank you! Questions?