Composite Extension Fields for (Network) Coding: Designs and Opportunities

Daniel E. Lucani
Aalborg University
del@es.aau.dk

Joint work with J. Heide, O. Geil, D. Ruano
N. Hernandez, C. W. Soerensen

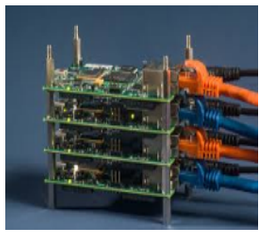
NC for Communications

A network revolution: replace store and forward with code and forward

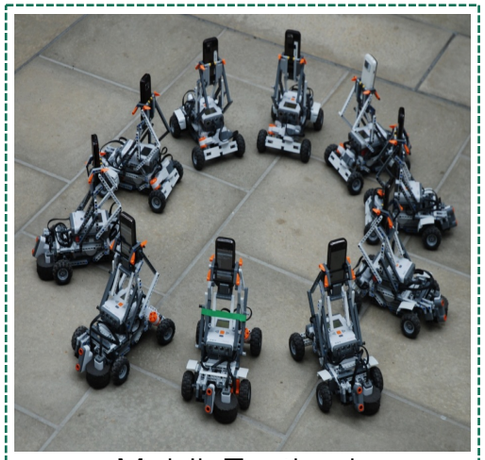
- Concept: Representing and manipulating data as equations in the network
- Focus:
  - Computational complexity
  - Protocol design
  - Latency
  - Energy efficiency
  - Reliability
- Applications: D2D, multi-path mesh routing, reliable multicast, multimedia streaming, SDN
- Spin-off: steinwurf

Raspberry Pi Testbed

NS3 simulator extensions



Manycore Testbed



MobileTestbed

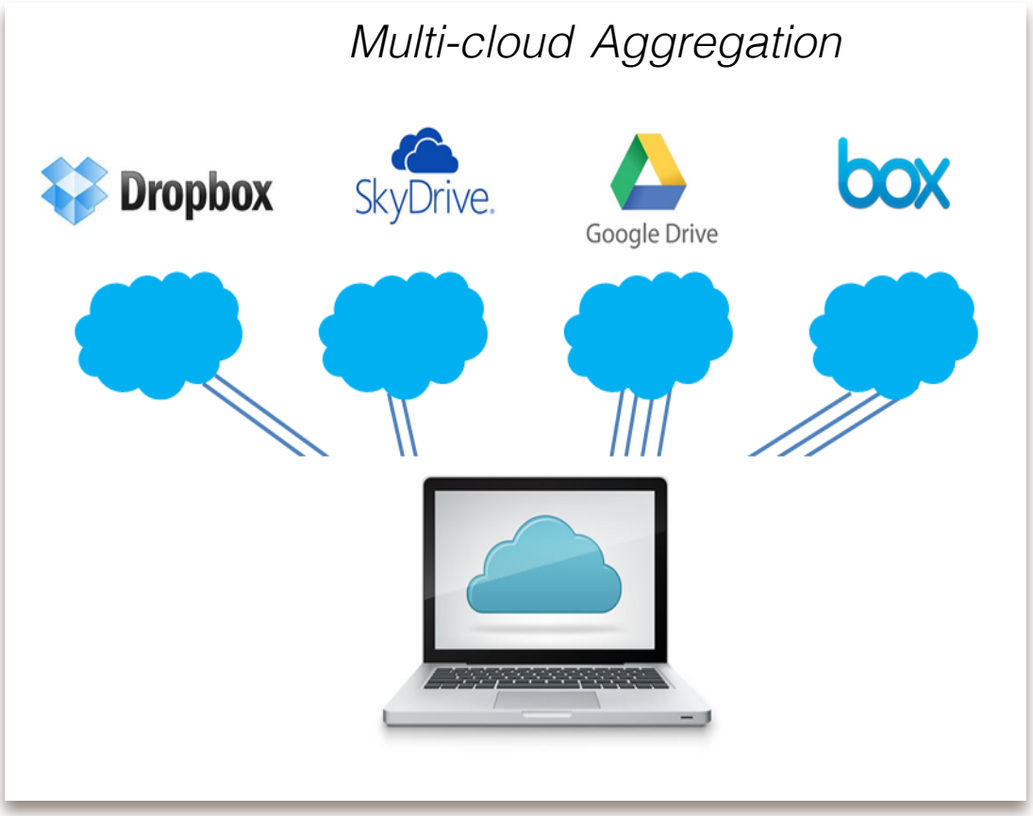
Smartphone Testbed

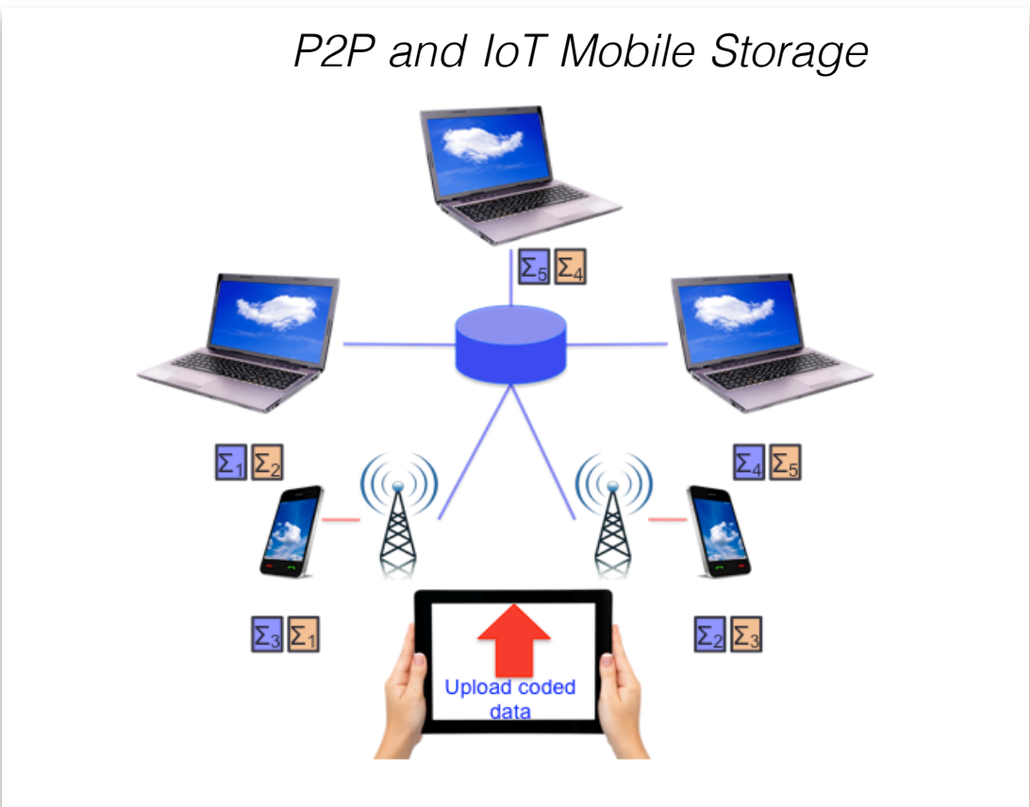
SDN Testbed

Network Coding for (Mobile) Storage

Dynamic storage need flexible codes

- Higher availability and security in existing cloud storage systems
  - Spreading data across several clouds
  - The code is the cipher
- Designing regenerating codes for data-center storage systems
  - Efficient local repair
  - Hierarchical storage (data-center, rack, node, disk)
- Designing codes for mobile, P2P, and IoT enabled storage
- Spin-off: Chocolate Cloud

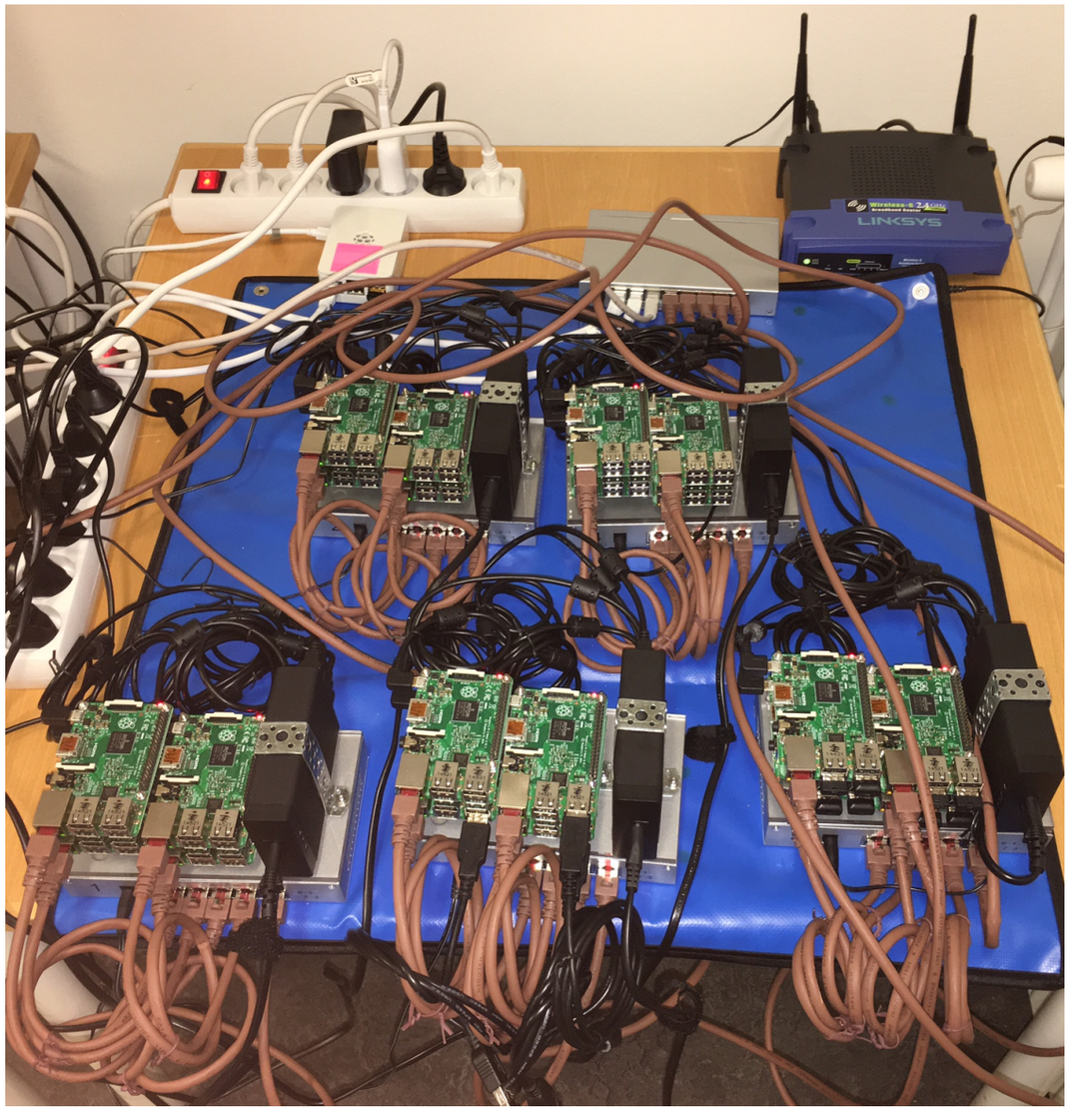




Network Coding for (Mobile) Storage

Dynamic storage need flexible codes

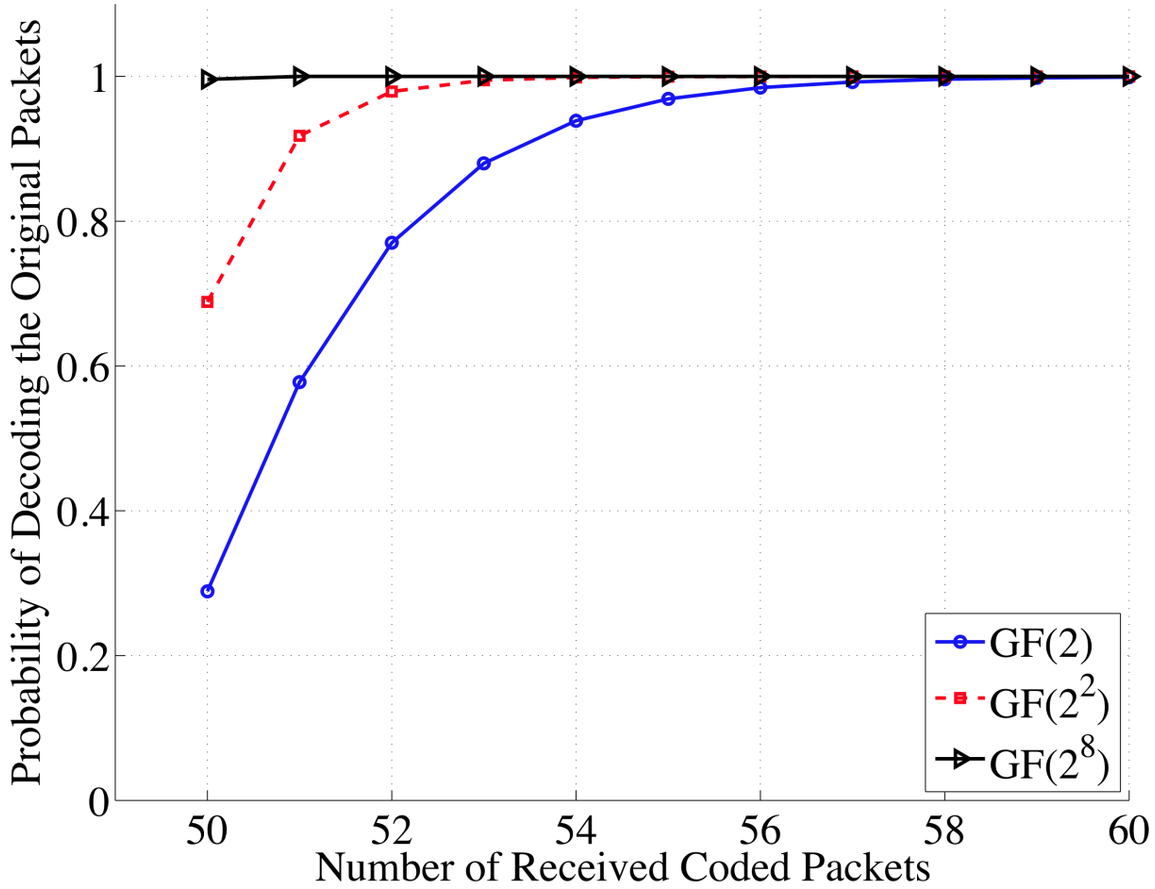
- Higher availability and security in existing cloud storage systems
  - Spreading data across several clouds
  - The code is the cipher
- Designing regenerating codes for data-center storage systems
  - Efficient local repair
  - Hierarchical storage (data-center, rack, node, disk)
- Designing codes for mobile, P2P, and IoT enabled storage
- Spin-off: Chocolate Cloud

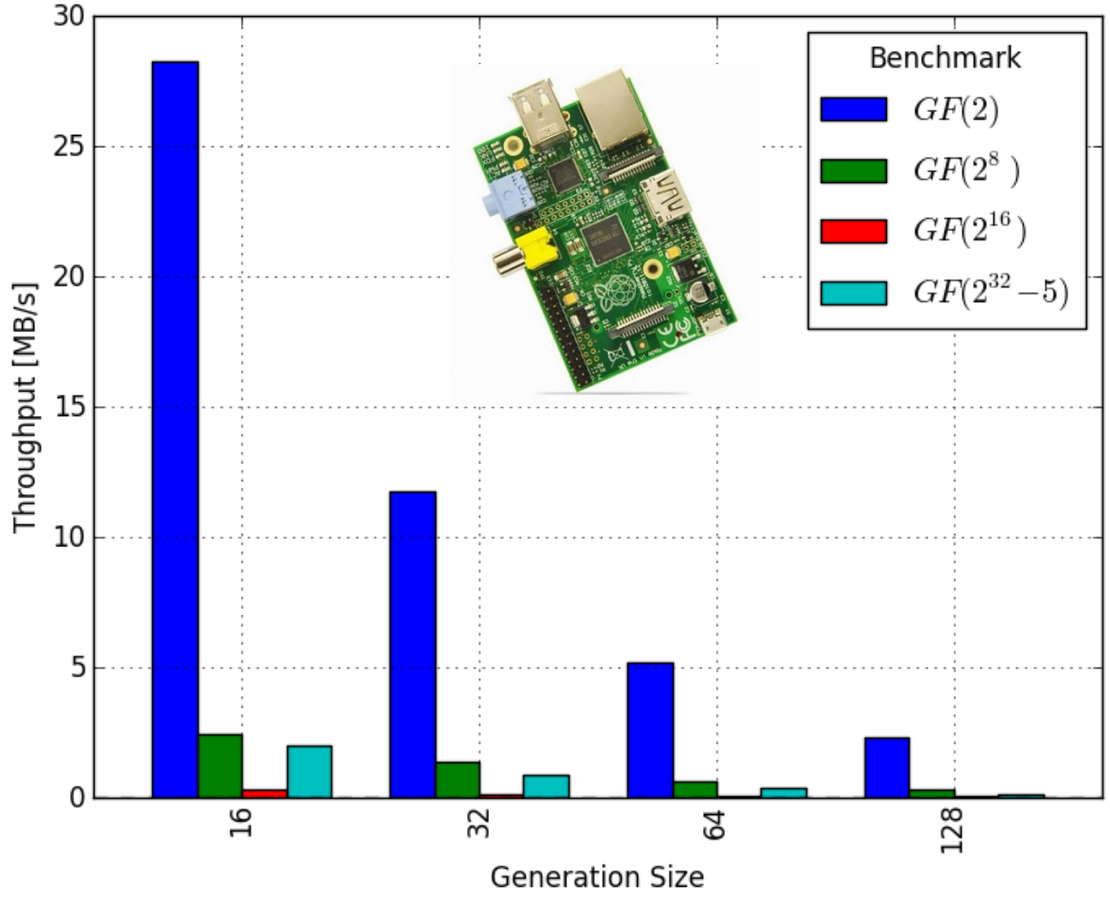


Introduction

- Network coding is a promising paradigm
  - Store-code-forward instead of store-forward
  - Benefits in throughput, delay, robustness, energy
- Caveats:
  - Computational complexity
  - Overhead per packet, e.g., signaling of coding coefficients

Motivation





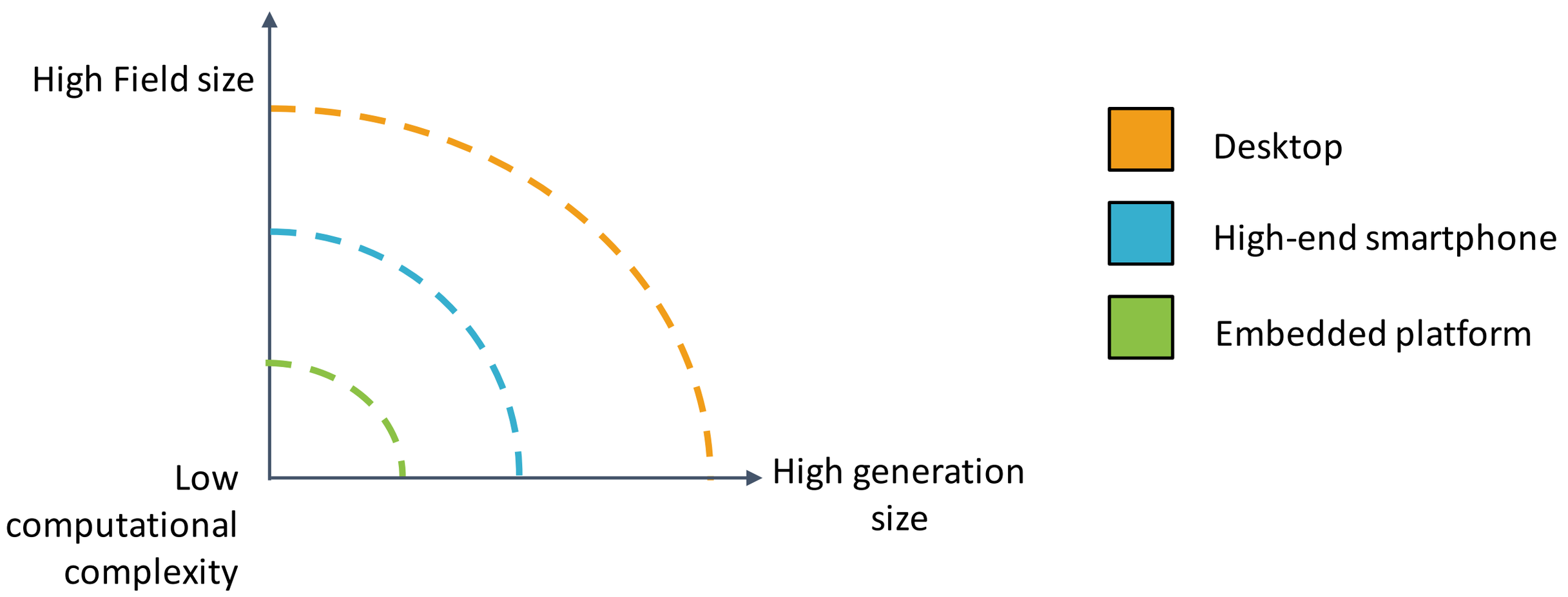
Larger Fields
are more efficient

?

Smaller Fields
are faster and have
less overhead

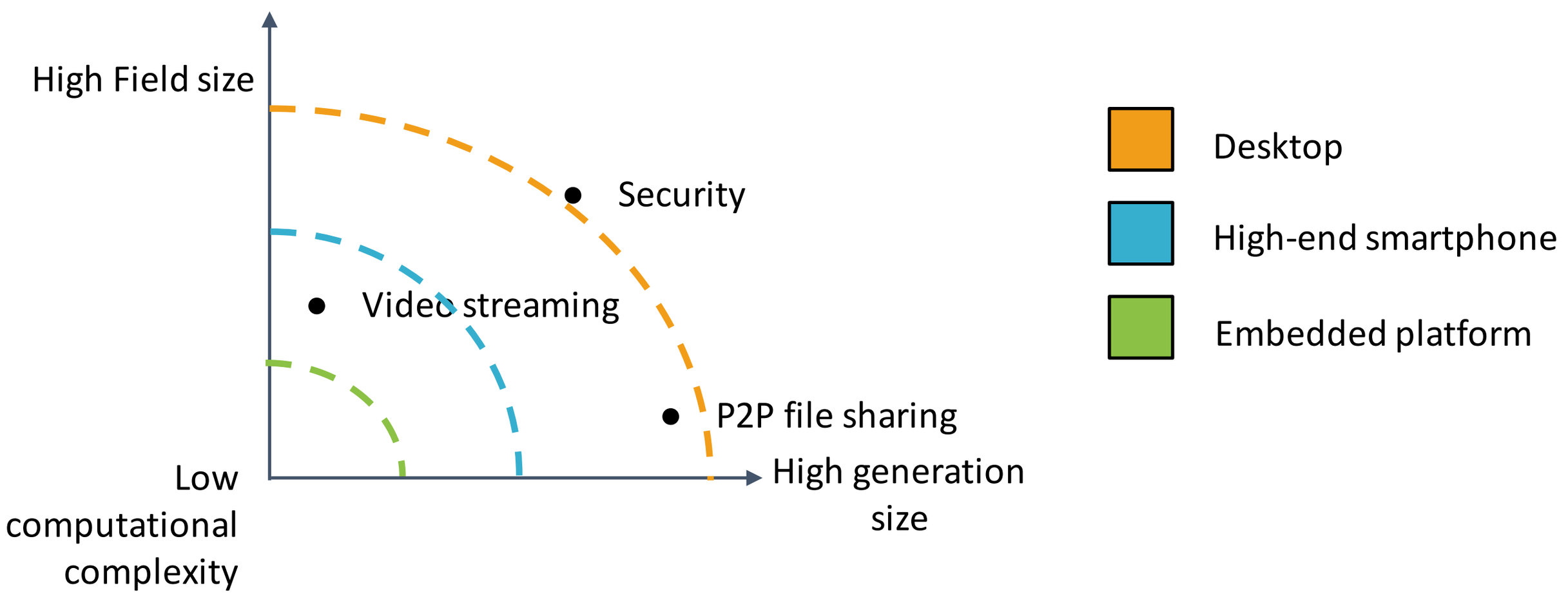
Overhead and Complexity

- We need to choose a field size and generation size (window size)



Overhead and Complexity

- We need to choose a field size and generation size (window size)
- Applications, choice of field, requirements: may change



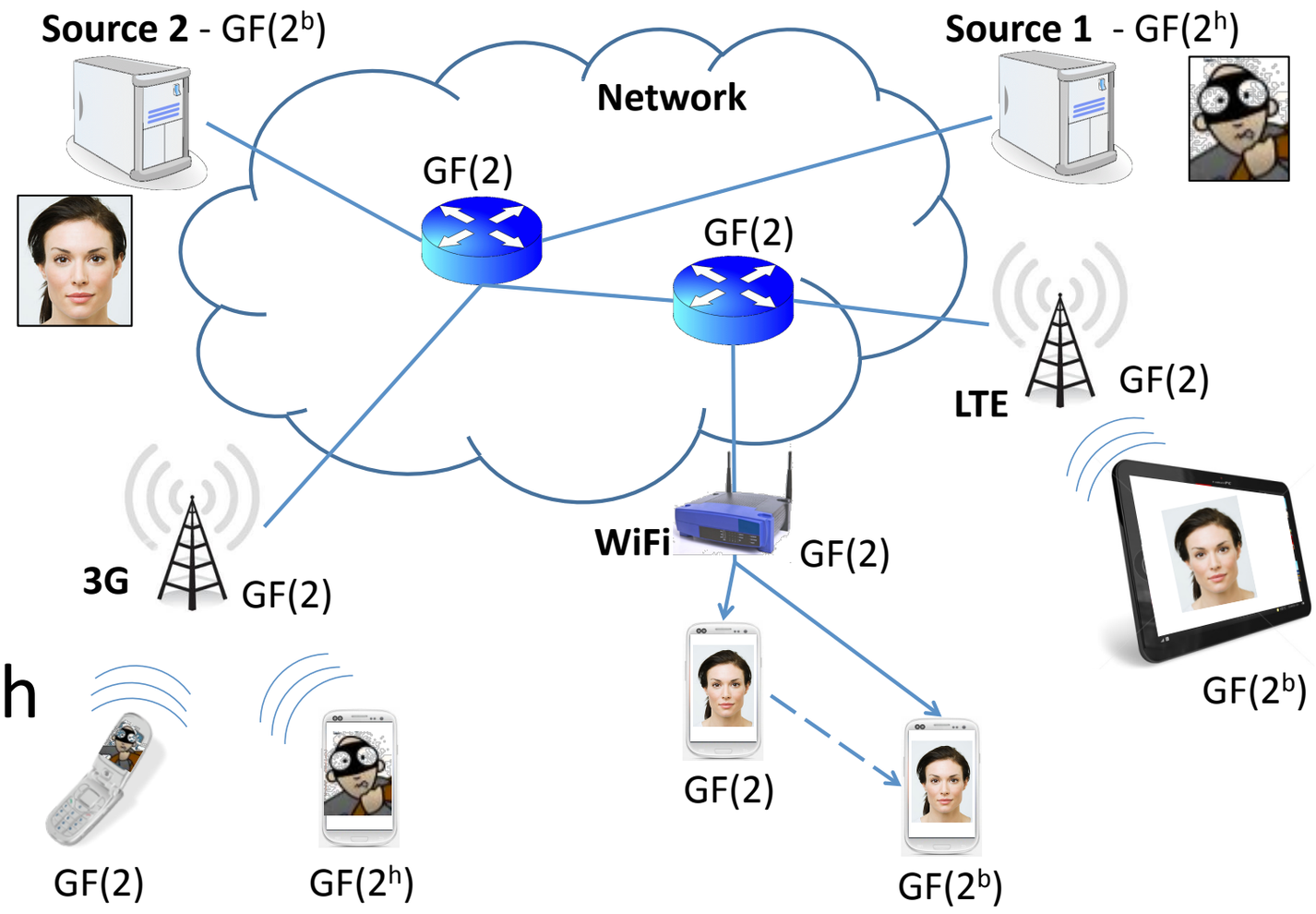
Introduction (revisited)

- Network coding is a promising paradigm
  - Store-code-forward instead of store-forward
  - Benefits in throughput, delay, robustness, energy
- Caveats:
  - Computational complexity
  - Overhead per packet, e.g., signaling of coding coefficients
- **Previous work:**
  - Understanding single choice of field size, generation size on processing speed [Paramanathan et al 2013]
  - Techniques that exploit sparsity: tunable sparse network coding, overlapping generations, BATS codes, ...
  - Fulcrum network codes: low overhead, high processing speed, simple recoding, configurable performance [Lucani et al 2014]

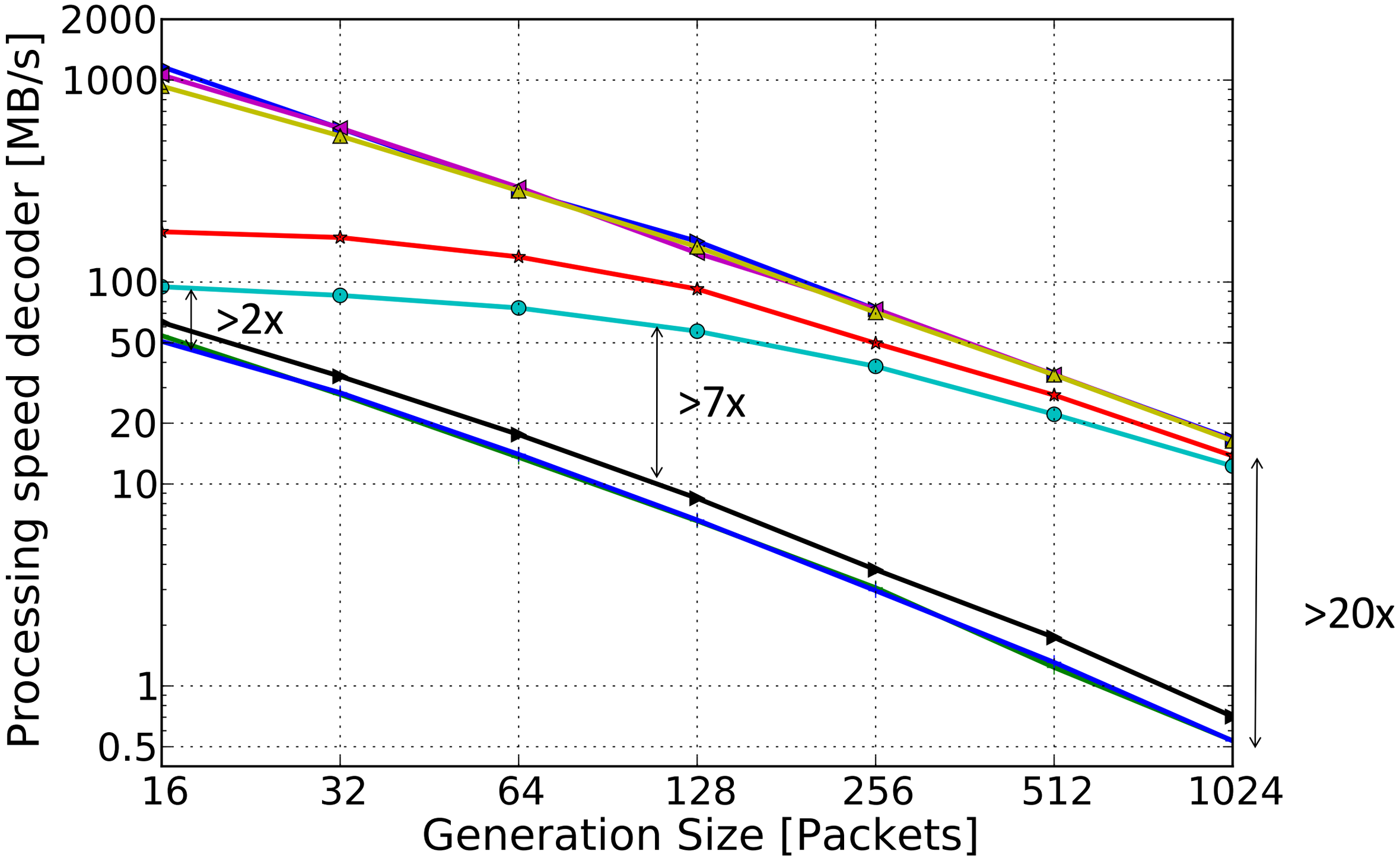
Fulcrum network codes (highlights)

- Fluid allocation of complexity
- End devices agree on desired performance:
- Independent from network
- Chosen according to application requirements
- Network devices need only support a simple subset of functions
- Reduces overhead
- Roughly 1 bit per coding coefficient

Key: code concatenation with different field sizes



Performance Results: Decoder



Motivation

Fulcrum provides support using composite fields:

GF(2) in the network and any GF(2^k) at source and destinations

Key: operations performed in GF(2) have an equivalent operation in any GF(2^k)

Example: $P_1 + P_2$ requires a bit by bit XOR in both cases

Motivation

Fulcrum provides support using composite fields:

GF(2) in the network and any GF(2^k) at source and destinations

Key: operations performed in GF(2) have an equivalent operation in any GF(2^k)

Example: $P_1 + P_2$ requires a bit by bit XOR in both cases

Can we expand this idea for more flexibility in code design?

GF(2^8) GF(2^8) GF(2^8)

Motivation

Fulcrum provides support using composite fields:

GF(2) in the network and any GF(2^k) at source and destinations

Key: operations performed in GF(2) have an equivalent operation in any GF(2^k)

Example: $P_1 + P_2$ requires a bit by bit XOR in both cases

Can we expand this idea for more flexibility in code design?

GF(2^8) → GF(2^2) → GF(2^8)

Motivation

Fulcrum provides support using composite fields:

GF(2) in the network and any GF(2^k) at source and destinations

Key: operations performed in GF(2) have an equivalent operation in any GF(2^k)

Example: $P_1 + P_2$ requires a bit by bit XOR in both cases

Can we expand this idea for more flexibility in code design?

Typically, GF(2^k) is not *compatible* with a GF(2^p) if n ≠ p

Different primitive polynomials → different mapping for product operations

Motivation

Typically, GF(2^k) is not *compatible* with a GF(2^p) if n ≠ p

Example: GF(2^2) versus GF(2^4)

GF(2^2)

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

≠

GF(2^4)

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	4	6	8	10	…	13
3	3	6	5	12	15	…	2
4	4	8	12	3	7	…	9
5	5	10	15	7	2	…	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	13	2	9	6	…	10

Motivation

Typically, GF(2^k) is not *compatible* with a GF(2^p) if n ≠ p

Example: GF(2^2) versus GF(2^4)

GF(2^2)

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

≠

GF(2^4)

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	4	6	8	10	…	13
3	3	6	5	12	15	…	2
4	4	8	12	3	7	…	9
5	5	10	15	7	2	…	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	13	2	9	6	…	10

Actually, we will need even stronger conditions

Idea: Multiple composite extension fields

Motivation

Typically, GF(2^k) is not *compatible* with a GF(2^p) if n ≠ p

Example: GF(2^2) versus GF(2^4)

GF(2^2)

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

≠

GF(2^4)

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	4	6	8	10	…	13
3	3	6	5	12	15	…	2
4	4	8	12	3	7	…	9
5	5	10	15	7	2	…	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	13	2	9	6	…	10

Actually, we will need even stronger conditions

Idea: Multiple composite extension fields

Multiple Composite Extension Fields

Use GF(2) to construct GF(2^2), use GF(2^2) to construct GF(2^{2^2})

GF(2^2)

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

=

GF(2^{2^2})

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	3	1	8		…	5
3	3	1	2	12		…	10
4	4	8	12	6		…	1
5	5					…	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	5	10	1		…	9

Full compliance: product by 1, 2, 3 in GF(2^{2^2}) (4 bits) Is equivalent to the product on two values of GF(2^2) (2 bits)

Multiple Composite Extension Fields

Use GF(2) to construct GF(2^2), use GF(2^2) to construct GF(2^{2^2})

GF(2^2)

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

=

GF(2^{2^2})

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	3	1	8		…	5
3	3	1	2	12		…	10
4	4	8	12	6		…	1
5	5					…	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	5	10	1		…	9

Full compliance: product by 1, 2, 3 in GF(2^{2^2}) (4 bits) Is equivalent to the product on two values of GF(2^2) (2 bits)

Multiple Composite Extension Fields

Use GF(2) to construct GF(2^2), use GF(2^2) to construct GF(2^{2^2})

Example: 15 x 2 = 5

1111_b x 0010_b in GF(2^{2^2})

But in GF(2^2) we operate on pairs of bits

11_b x 10_b = 01_b

Thus, on a vector

11_b11_bx 10_b = 01_b01_b

Which results in “5” for the larger field

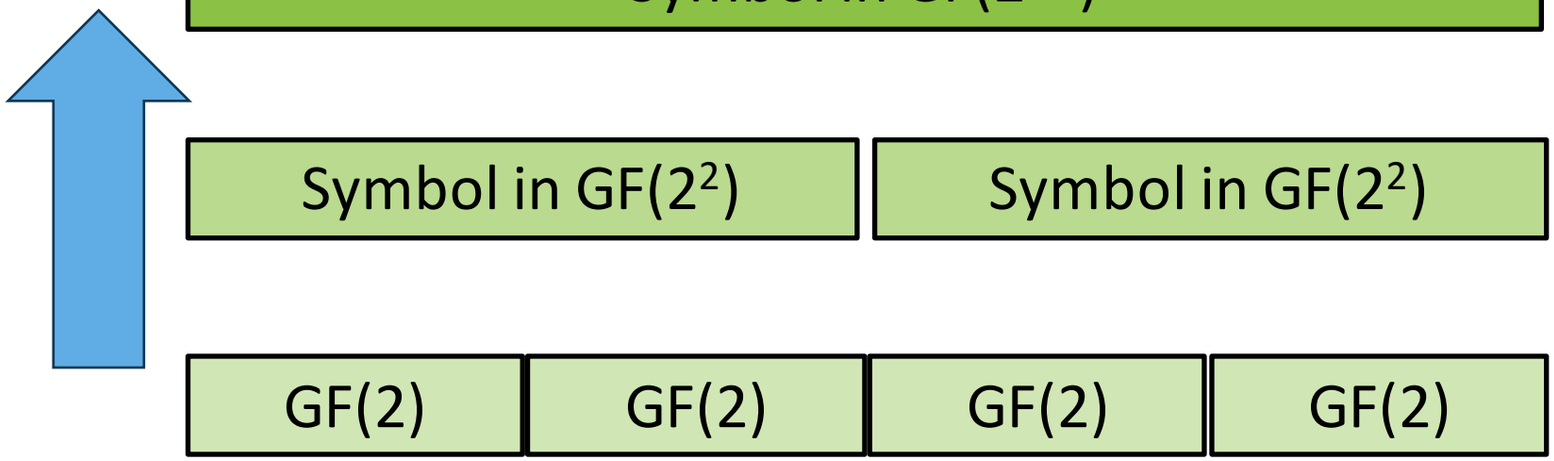
GF(2^{2^2})

x	1	2	3	4	5	…	15
1	1	2	3	4	5	…	15
2	2	3	1	8		…	5
3	3	1	2	12		…	10
4	4	8	12	6		…	1
5	5					…	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	15	5	10	1		…	9

Multiple Composite Extension Fields

1111_b x 0010_b = 0101_b

11_b11_bx 10_b = 01_b01_b



Finding the polynomials

- Simple if doubling bit count
  - Irreducible: $m(x) = x^2 + m_1 x + m_0 \neq (x + b1)(x + b2)$
  - Primitive: spans all values of the field
    - Long division of $x^k - 1$ modulo $m(x)$ for increasing integer k from $k = 1$
- Plenty of primitive polynomials
  - Field size of 16: 4
  - Field size 256: 64 after picking one primitive polynomial in field of size 16

Background

- Conway polynomials
- Use of composite extension fields to be able to process in large fields (security)

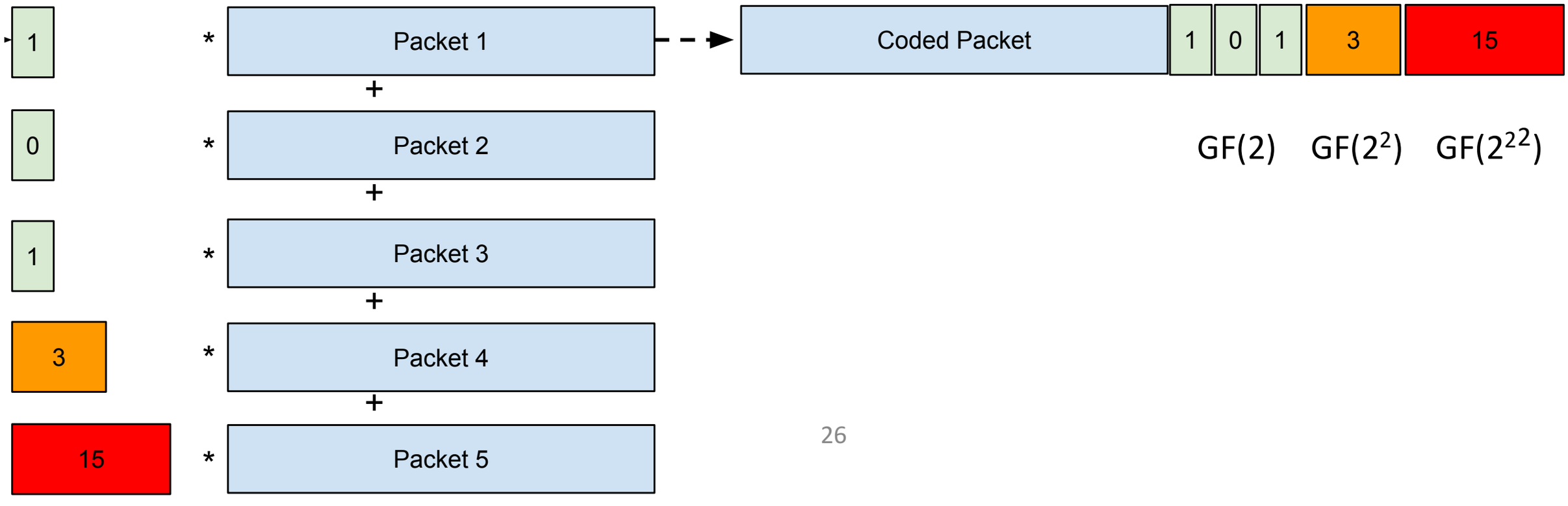
Designs and Opportunities

- Telescopic codes
- Network codes
- Fulcrum network codes
- Others

Telescopic Codes

Design:

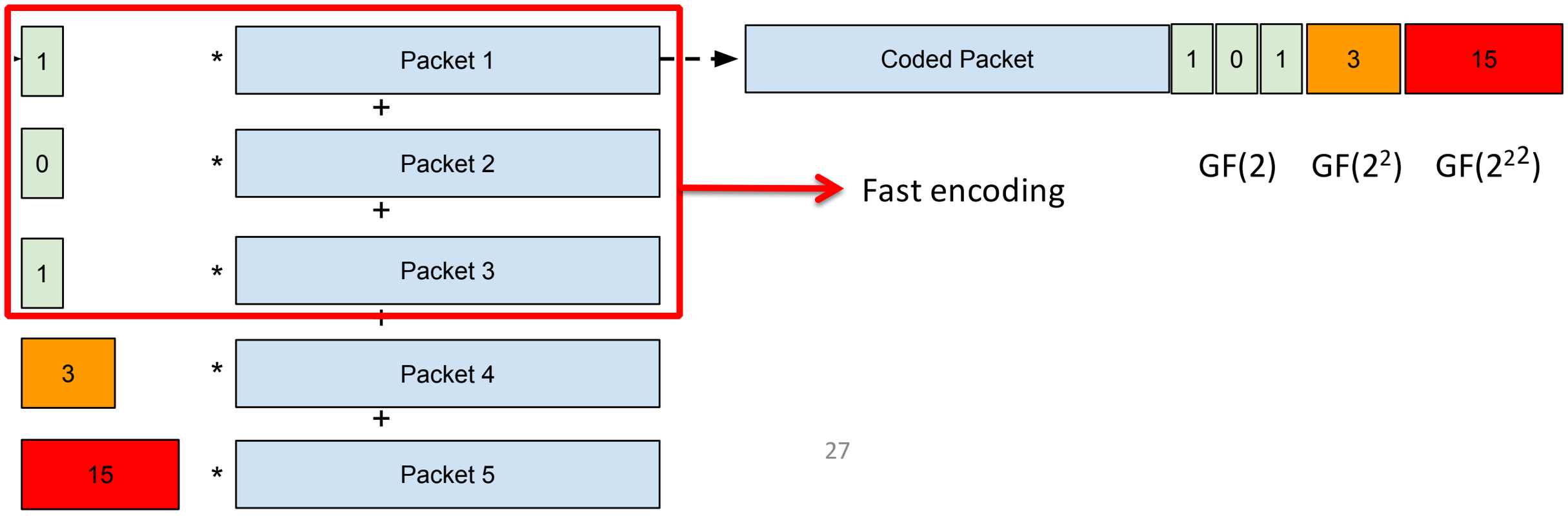
- Multiple composite extension fields
- Goal: reduce overhead, maintaining high performance, faster encoding/decoding
- Different packets are encoded using different field sizes



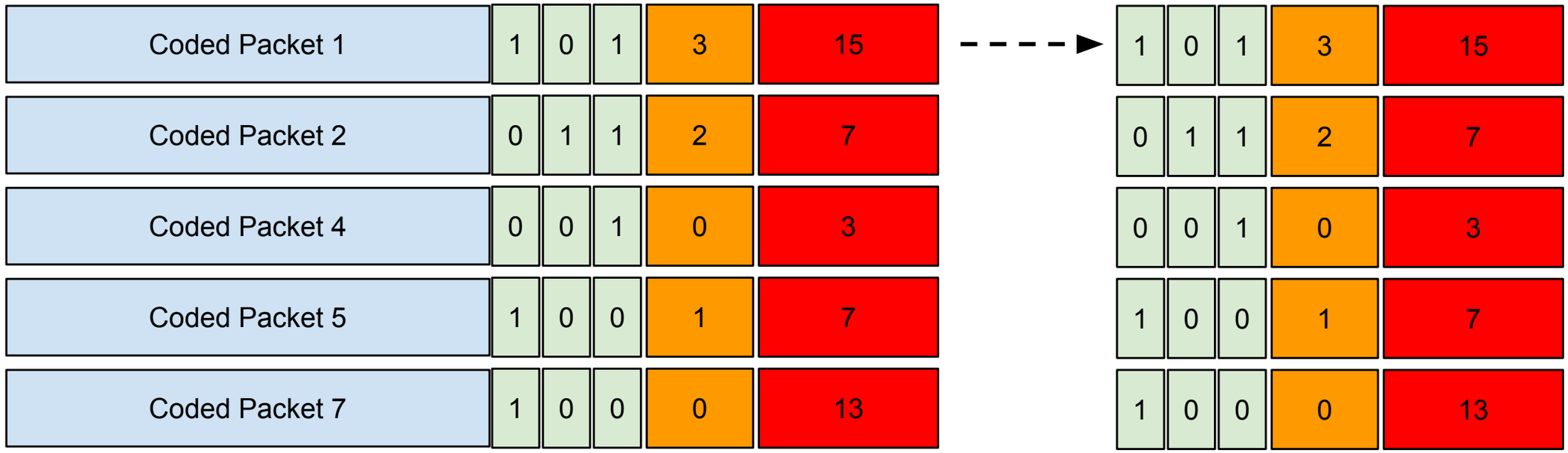
Telescopic Codes

Design:

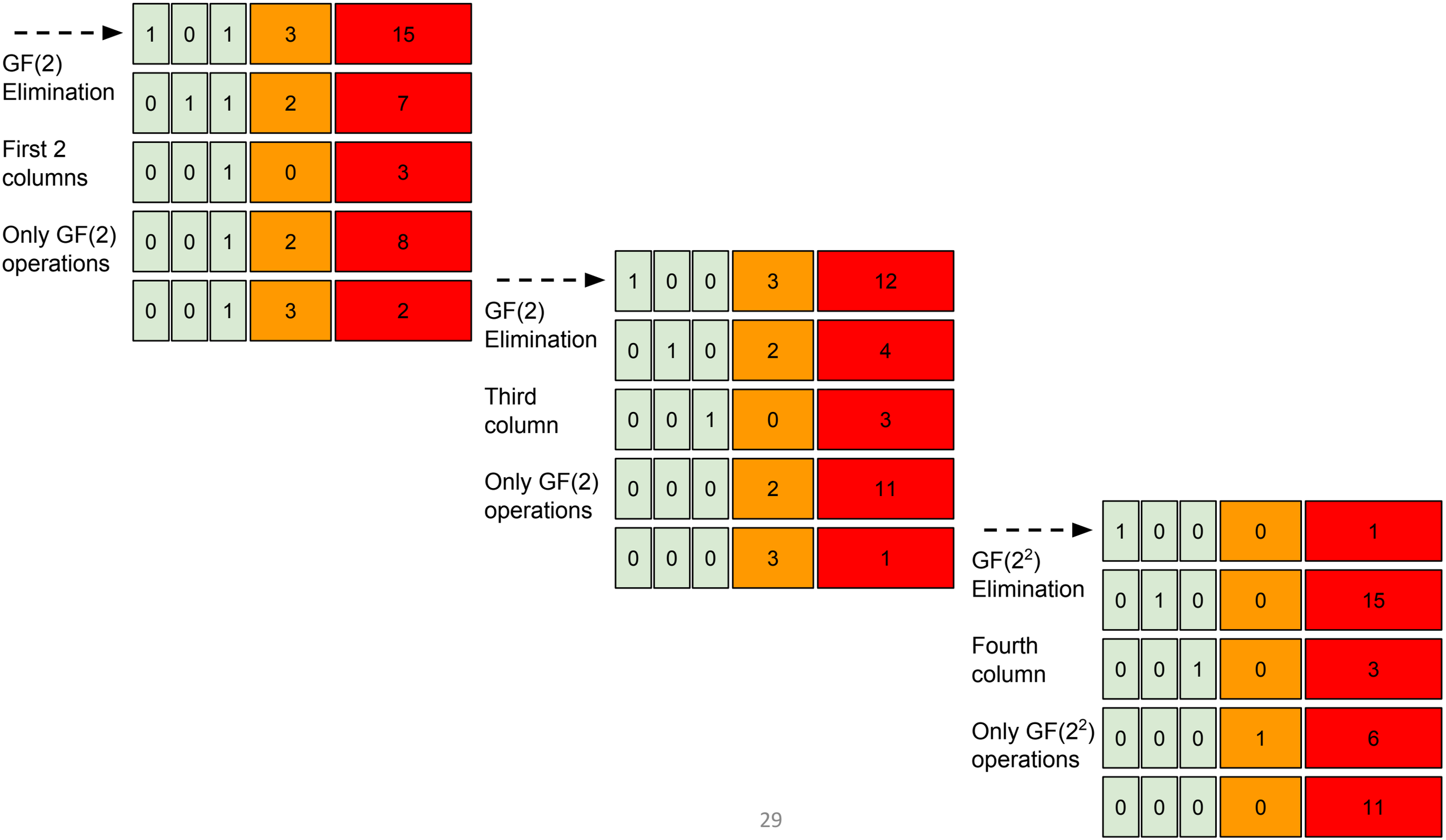
- Multiple composite extension fields
- Goal: reduce overhead, maintaining high performance, faster encoding/decoding
- Different packets are encoded using different field sizes



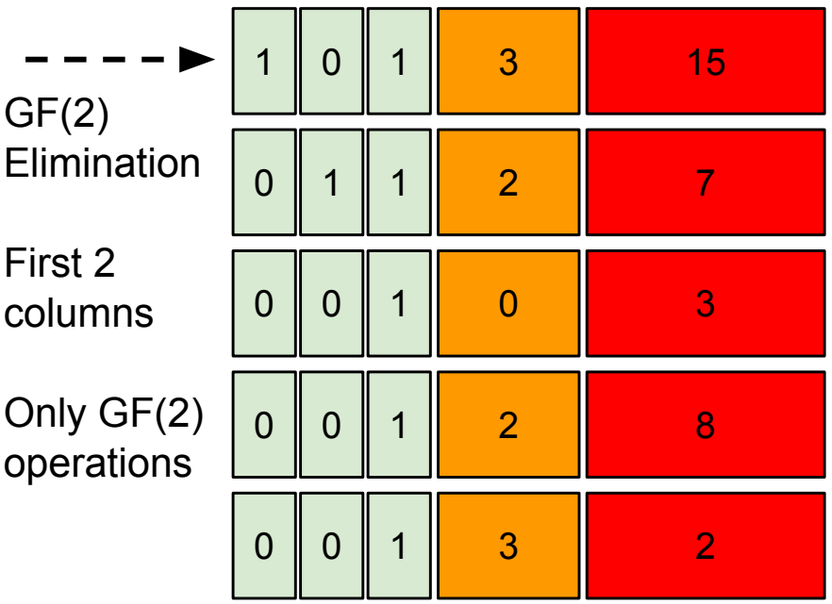
Telescopic Codes: Decoder

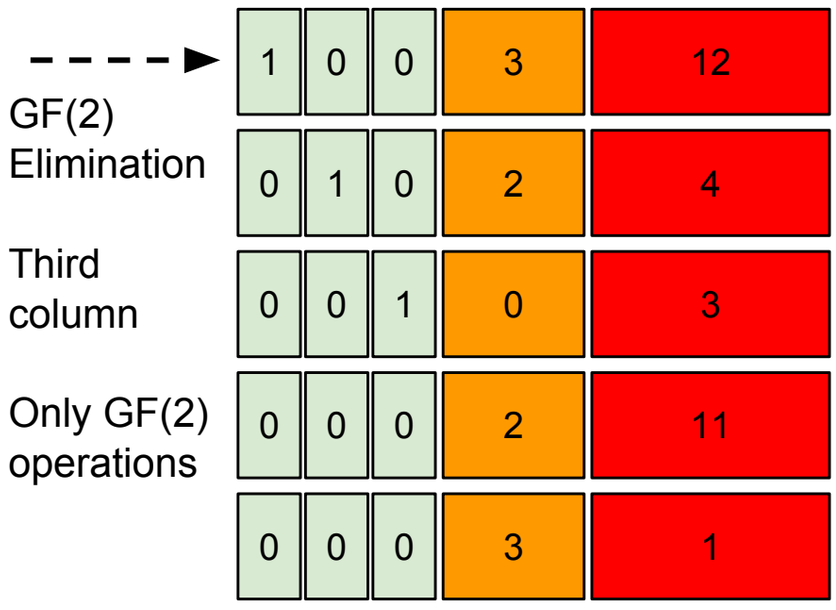


Telescopic Codes: Decoder

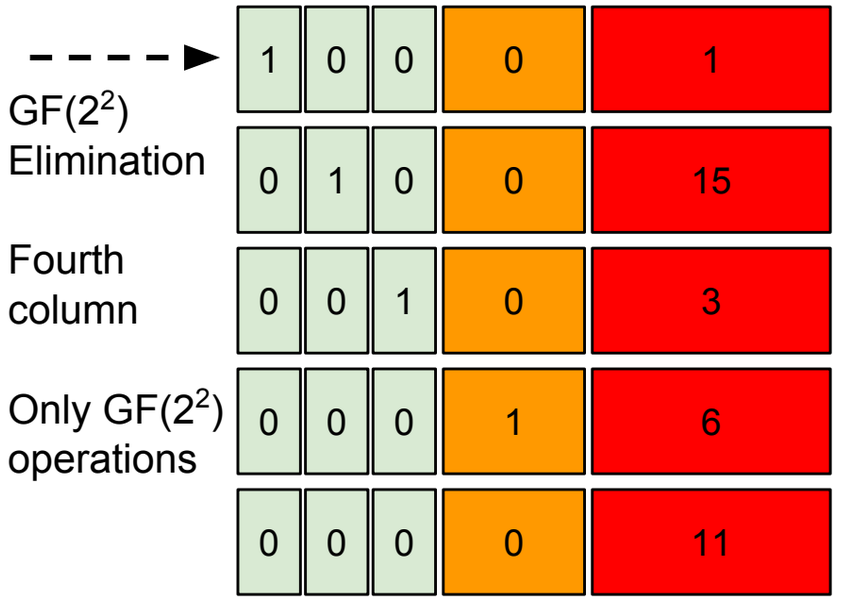


Telescopic Codes: Decoder





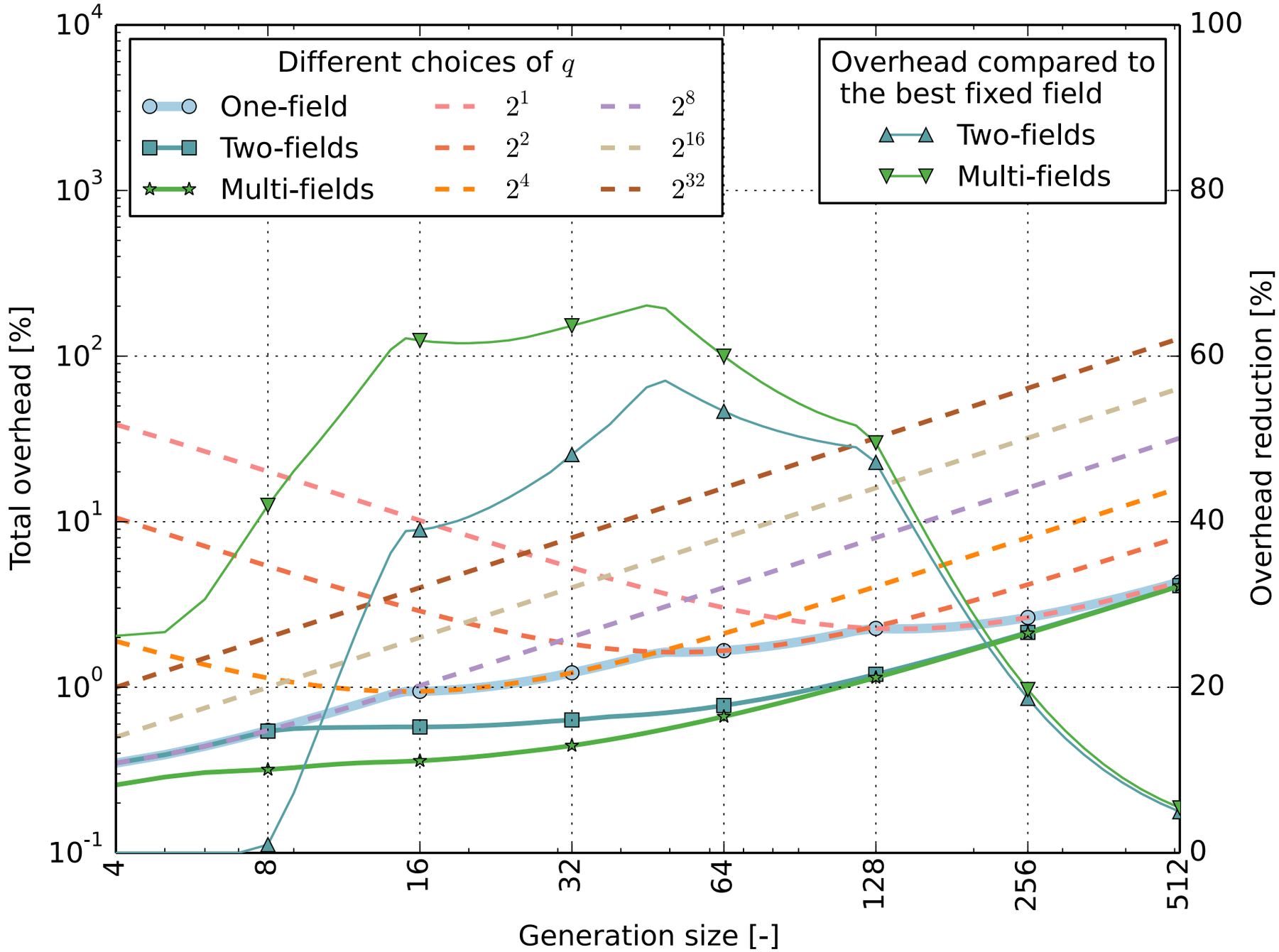
$11_d \times 3_d =$
$1011_b \times 11_b =$
$01_b10_b =$
6_d



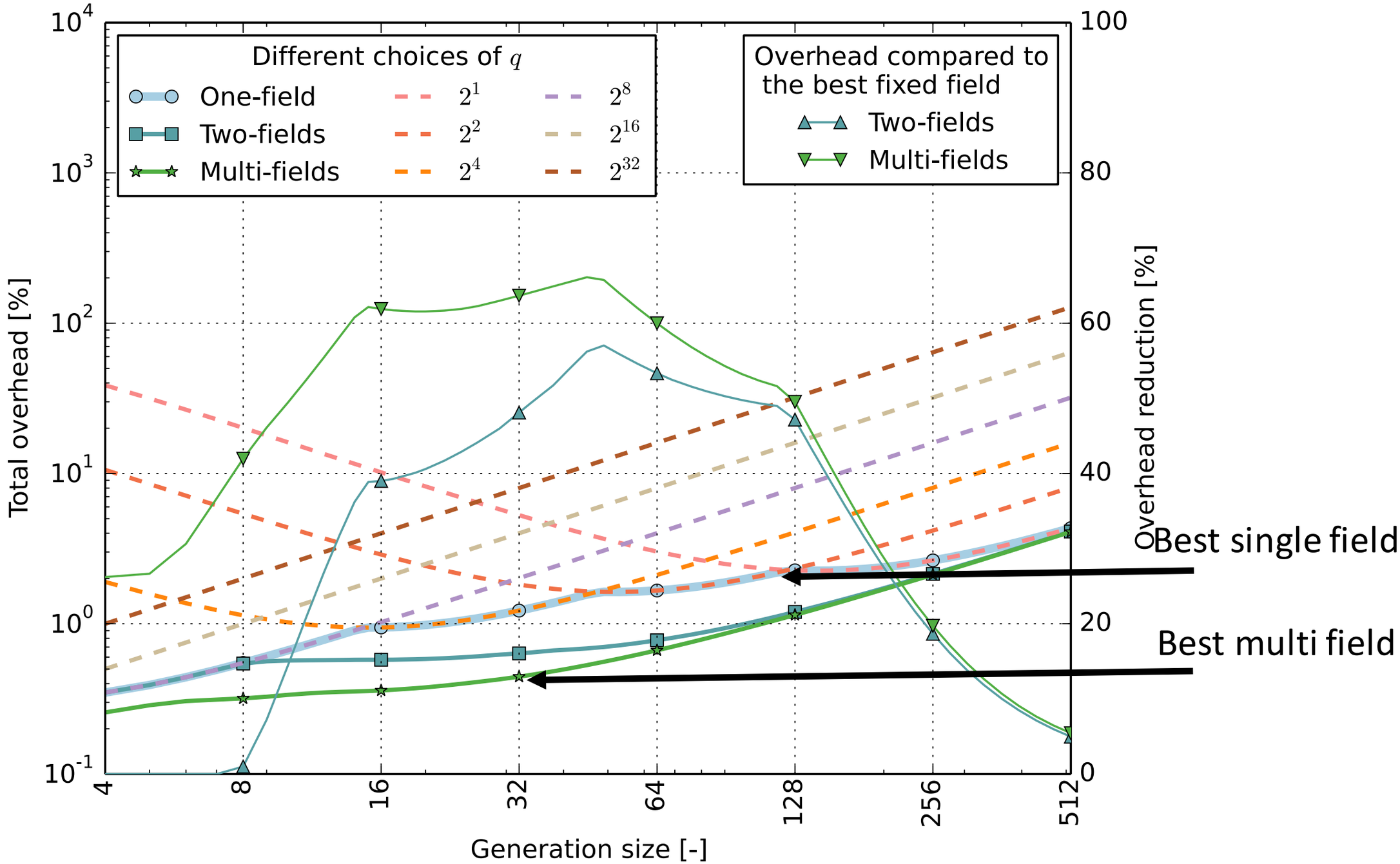
$GF(2^2)$

×	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

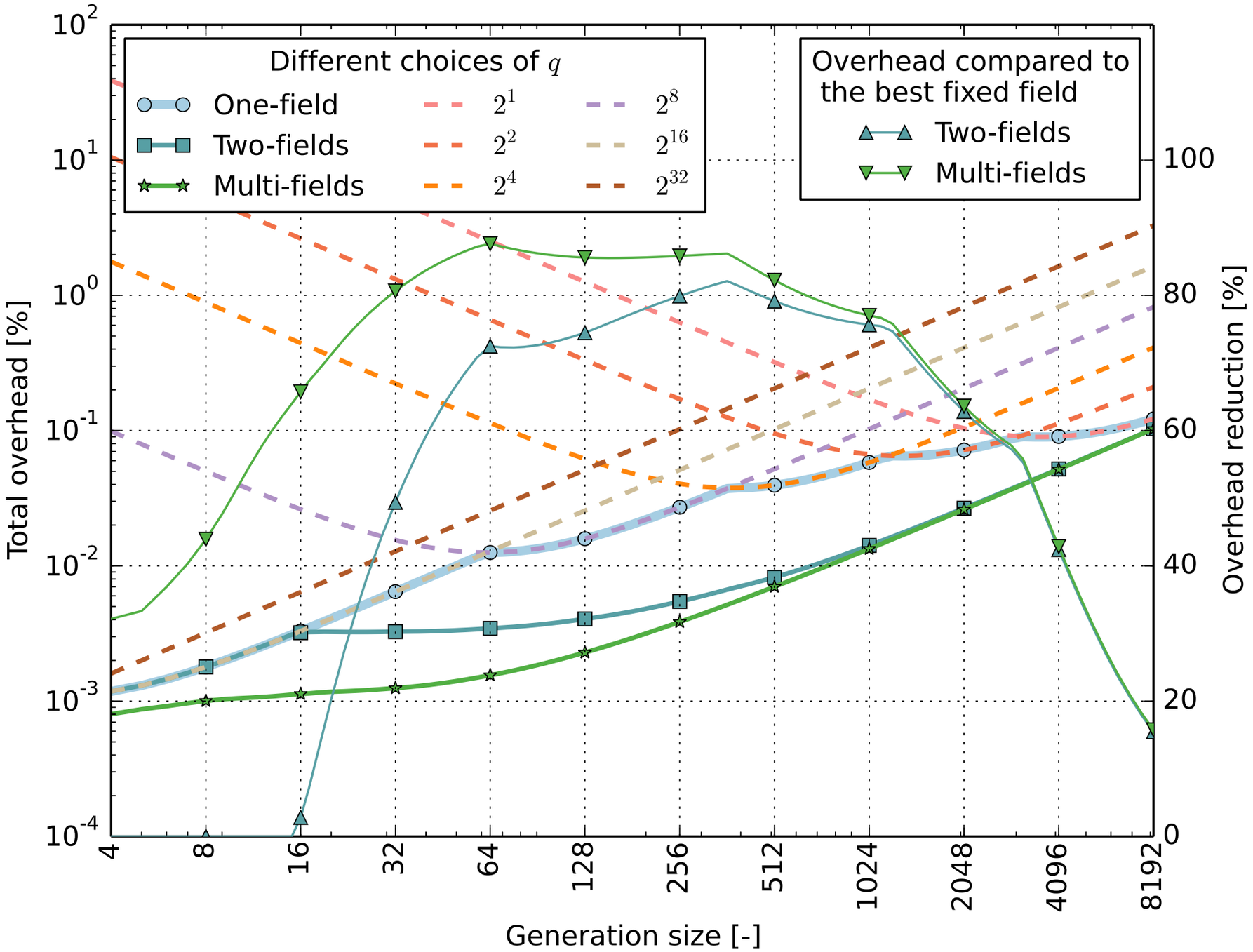
Results: Overhead (1600B pkts)



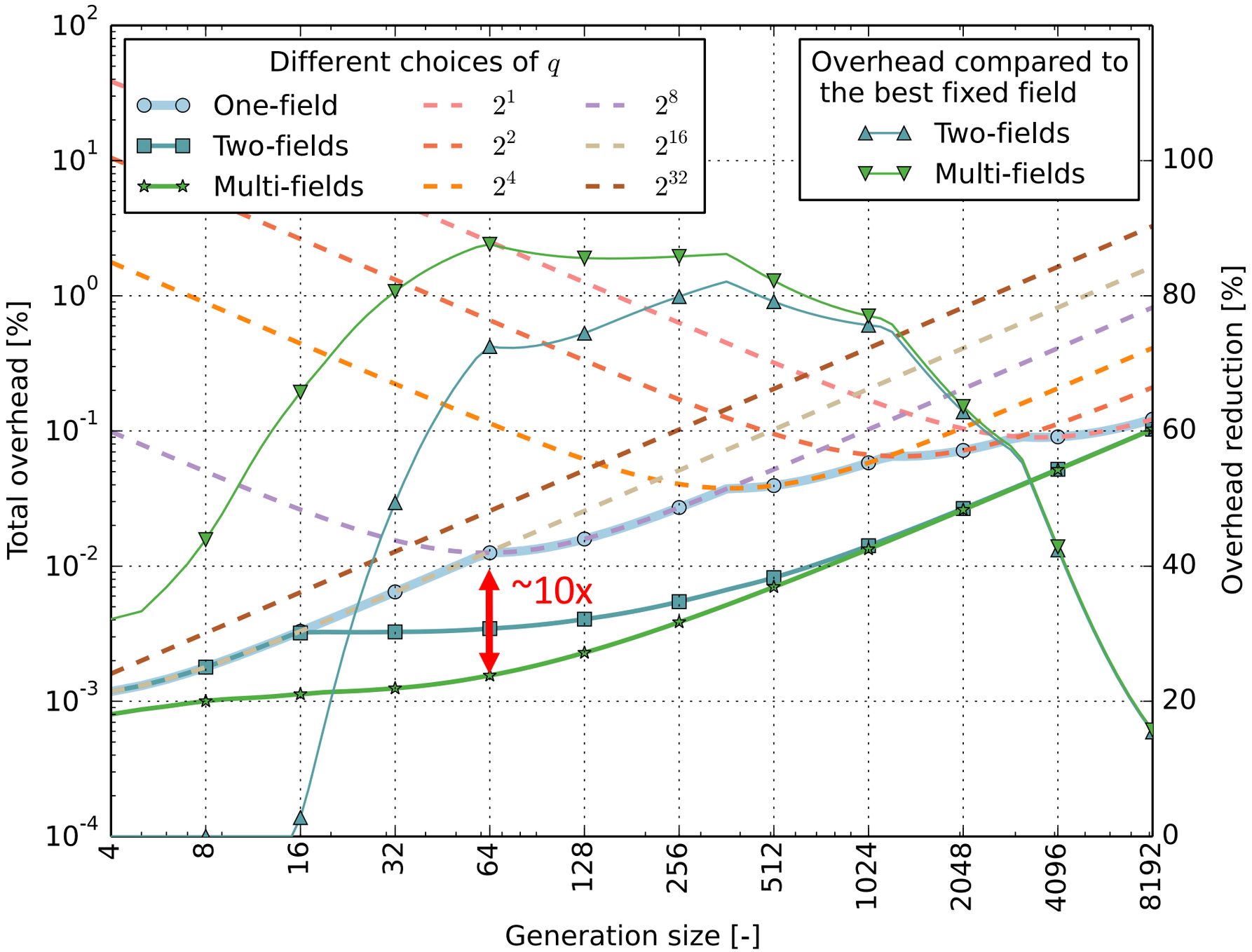
Results: Overhead (1600B pkts)



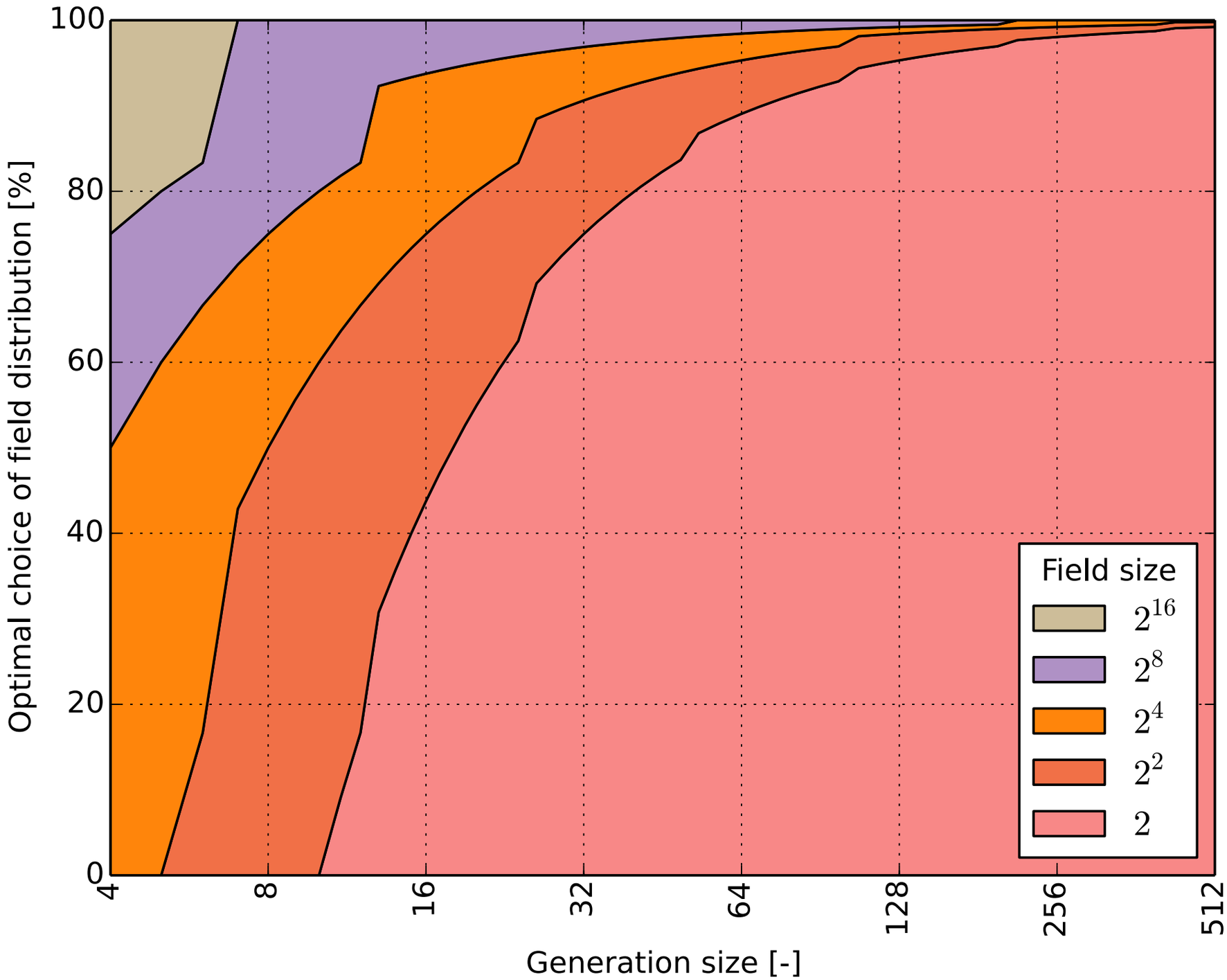
Results: Overhead (1MB pkts)



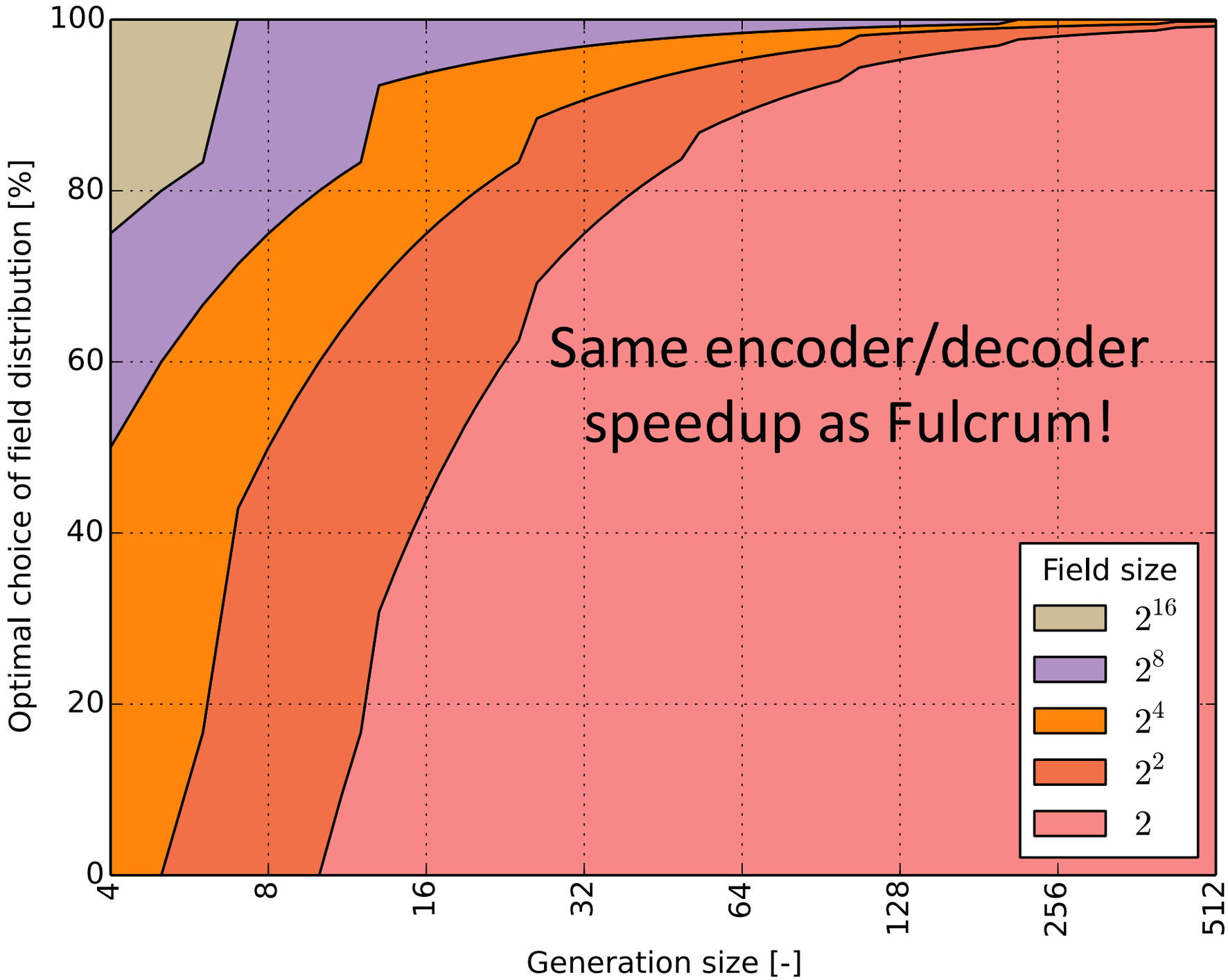
Results: Overhead (1MB pkts)



Results: Optimal Choice of Field Distribution

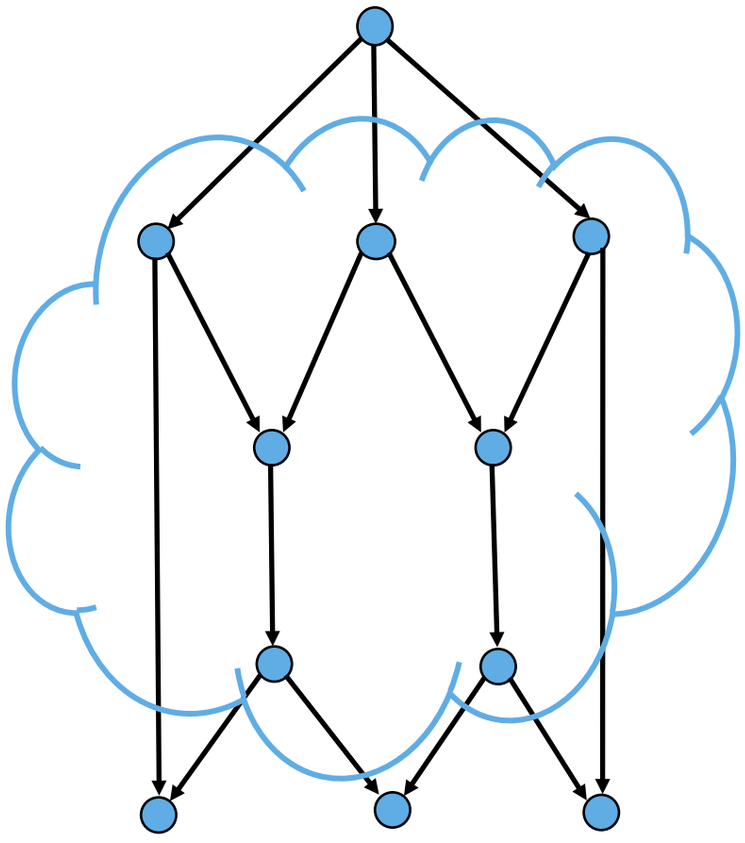


Results: Optimal Choice of Field Distribution



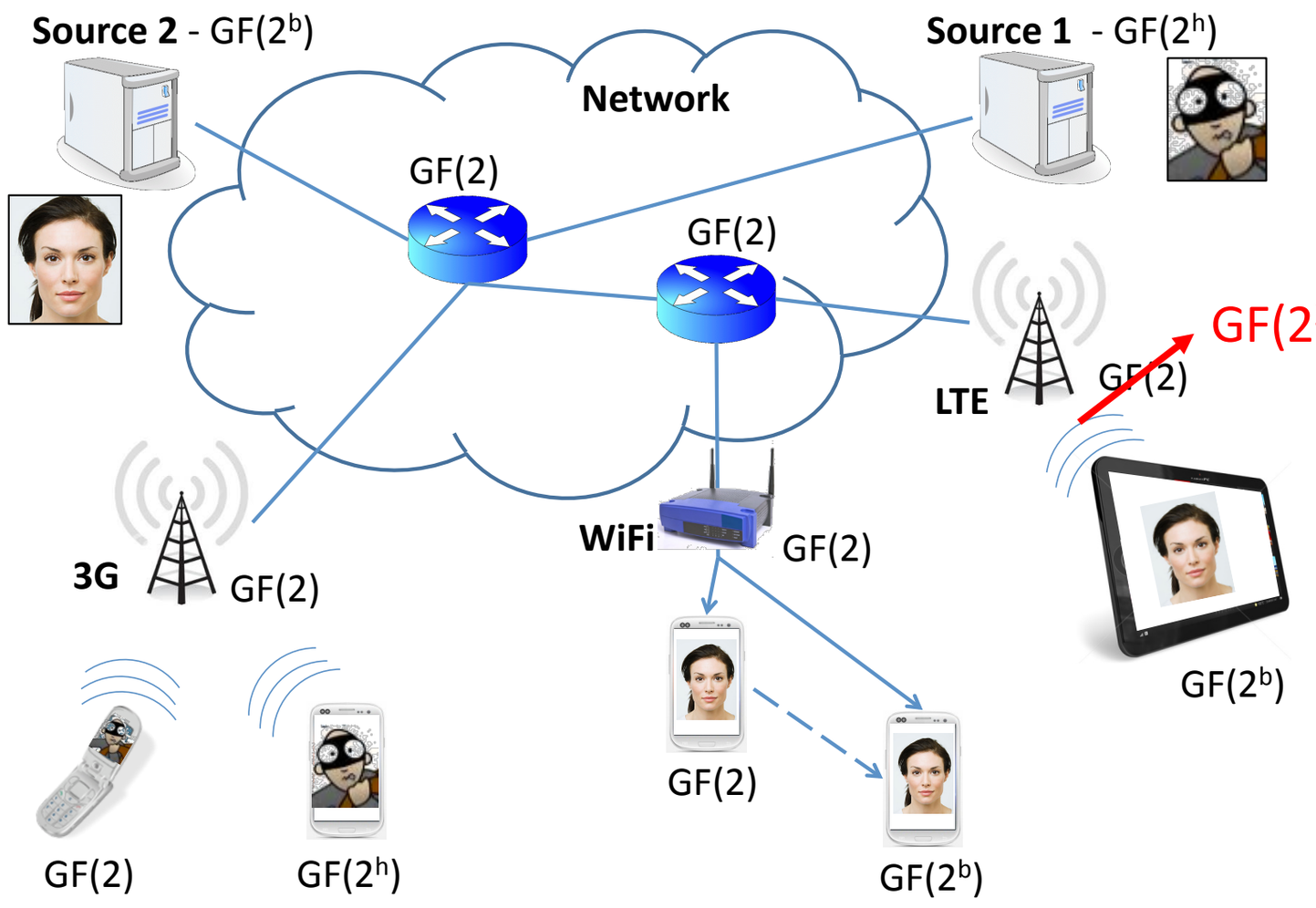
RLNC

- Allow intermediate nodes to opt for smaller coefficient values
- Reduce cost of processing
- Effects on decoding probability?



Fulcrum network codes

- Potential overhead increase
- Ex: Nodes in network can choose to recode with GF(2) or GF(2^2)
- More flexibility
- Maintain fast decoding algorithms



Open questions

- Can we re-think our families of known codes?
  - Early results with Olav Geil and Diego Ruano show it is possible (with some restrictions)
  - Preparing article on code construction
- Can we re-think code concatenation?
- Can we generate new tunable code constructions?

How C++ Combats Global Warming

Conclusions

- Simple design of composite extension fields
- Implications in code design
- Telescopic codes
  - Reduce total overhead
  - Maintain high decoding probability
  - Potential for processing at high speed (simpler encoder/decoder)
- Open questions to the community
  - Applications in other codes
  - New code designs
  - Performance evaluation in real systems