Network Coding (NC)

CITHN2002 – Summer 2024

Prof. Dr.-Ing. Stephan Günther

Chair of Distributed Systems and Security
School of Computation, Information and Technology
Technical University of Munich

Chapter 1: Random Linear Network Coding

Chapter 1: Random Linear Network Coding

Finite fields

Remember finite fields $\langle \mathbb{F}_q, +, \cdot \rangle$ with $q = p^n$ elements where $p \in \mathbb{N}$ is prime?

Finite fields

Remember finite fields $\langle \mathbb{F}_q, +, \cdot \rangle$ with $q = p^n$ elements where $p \in \mathbb{N}$ is prime?

Finite fields (Galois[1] fields)
Let p be a prime number and $n \in \mathbb{N}$ a natural number. Then, a finite field $\mathbb{F}_q$ is a set of $q = p^n$ elements and two binary operators $\langle +, \cdot \rangle$ such that $\langle \mathbb{F}, + \rangle$ and $\langle \mathbb{F} \backslash \{0\}, \cdot \rangle$ form Abelian groups with $\cdot$ being distributive over $+$.

[1] named after Évariste Golois, *1811 Bourg-la-Reine (France), †1832 in a duel

Finite fields

Remember finite fields $\langle \mathbb{F}_q, +, \cdot \rangle$ with $q = p^n$ elements where $p \in \mathbb{N}$ is prime?

Finite fields (Galois[1] fields)
Let p be a prime number and $n \in \mathbb{N}$ a natural number. Then, a finite field $\mathbb{F}_q$ is a set of $q = p^n$ elements and two binary operators $\langle +, \cdot \rangle$ such that $\langle \mathbb{F}, + \rangle$ and $\langle \mathbb{F} \setminus \{0\}, \cdot \rangle$ form Abelian groups with $\cdot$ being distributive over $+$.

Abelian[2] group
An Abelian group is a set of elements that fulfills the axioms of closure, identity, associativity, invertibility, and commutativity (the last axiom is specific to Abelian groups).

[1] named after Évariste Golois, *1811 Bourg-la-Reine (France), †1832 in a duel
[2] named after Niels Henrik Abel, *1802 Nedstrand (Norway), †1829 of pulmonary tuberculosis

Finite fields

Remember finite fields $\langle \mathbb{F}_q, +, \cdot \rangle$ with $q = p^n$ elements where $p \in \mathbb{N}$ is prime?

Finite fields (Galois[1] fields)
Let p be a prime number and $n \in \mathbb{N}$ a natural number. Then, a finite field $\mathbb{F}_q$ is a set of $q = p^n$ elements and two binary operators $\langle +, \cdot \rangle$ such that $\langle \mathbb{F}, + \rangle$ and $\langle \mathbb{F} \setminus \{0\}, \cdot \rangle$ form Abelian groups with $\cdot$ being distributive over $+$.

Abelian[2] group
An Abelian group is a set of elements that fulfills the axioms of closure, identity, associativity, invertibility, and commutativity (the last axiom is specific to Abelian groups).

If $q = p$, and we define for any $a, b \in \mathbb{F}_p$

$$a +_p b = (a + b) \bmod p, \text{ and}$$
$$a \cdot_p b = (a \cdot b) \bmod p,$$

then $\langle \mathbb{Z}_p, +_p, \cdot_p \rangle$ is a finite field.

[1] named after Évariste Golois, *1811 Bourg-la-Reine (France), †1832 in a duel
[2] named after Niels Henrik Abel, *1802 Nedstrand (Norway), †1829 of pulmonary tuberculosis

Finite fields

Remember finite fields $\langle \mathbb{F}_q, +, \cdot \rangle$ with $q = p^n$ elements where $p \in \mathbb{N}$ is prime?

Finite fields (Galois[1] fields)
Let p be a prime number and $n \in \mathbb{N}$ a natural number. Then, a finite field $\mathbb{F}_q$ is a set of $q = p^n$ elements and two binary operators $\langle +, \cdot \rangle$ such that $\langle \mathbb{F}, + \rangle$ and $\langle \mathbb{F} \setminus \{0\}, \cdot \rangle$ form Abelian groups with $\cdot$ being distributive over $+$.

Abelian[2] group
An Abelian group is a set of elements that fulfills the axioms of closure, identity, associativity, invertibility, and commutativity (the last axiom is specific to Abelian groups).

If $q = p$, and we define for any $a, b \in \mathbb{F}_p$

$$a +_p b = (a + b) \bmod p, \text{ and}$$
$$a \cdot_p b = (a \cdot b) \bmod p,$$

then $\langle \mathbb{Z}_p, +_p, \cdot_p \rangle$ is a finite field.

Note:

- $\langle \mathbb{Z}_4, +_4, \cdot_4 \rangle$ is **not** a finite field as 4 is obviously not prime.
- However, there is a finite field with 4 elements since 4 is a prime power. We just have to define addition and multiplication accordingly.

[1] named after Évariste Golois, *1811 Bourg-la-Reine (France), †1832 in a duel
[2] named after Niels Henrik Abel, *1802 Nedstrand (Norway), †1829 of pulmonary tuberculosis

Finite fields

Example: the binary field $\langle \mathbb{F}_2, +, \cdot \rangle$

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

$\Rightarrow$ addition is a bitwise XOR, multiplication a bitwise AND.

- Cool for computers since those machines tend to work with bits.
- Even cooler if it was possible to work on their native block size, which is a Byte[3] of 8 bit.
- However, $q = 256$ with $+$ and $\cdot$ defined modulo-style is not a finite field. :(
- But wait: $q = 2^8$ is a prime power, and our definition claimed that there are finite fields for any p^n as long as p is prime. :)
- For the same reason there is also a finite field with 4 elements ...

[3] Fun fact: the size of a Byte is **not** defined. We merely assume it is a block of 8 bit since there are few computers around that work on 7 bit blocks. It may be more correct to speak of word size, which simply depends on the architecture. By the way, Intel defines a word as 2 B although x86 architectures allow for byte-wise addressing of memory.

Binary extension fields

A binary extension field is a set of polynomials

$$F_q[x] = \left\{ \sum_{i=0}^{n-1} a_i x^i \;\middle|\; a_i \in \mathbb{F}_2 \right\}$$

of order $q = 2^n$ with appropriately defined binary operators $\langle +, \cdot \rangle$.

Binary extension fields

A binary extension field is a set of polynomials

$$F_q[x] = \left\{ \sum_{i=0}^{n-1} a_i x^i \;\middle|\; a_i \in \mathbb{F}_2 \right\}$$

of order $q = 2^n$ with appropriately defined binary operators $\langle +, \cdot \rangle$.

Examples:

- $q = 2 \Rightarrow F_2[x] = \{0, 1\}$ (same as $\mathbb{F}_2$)
- $q = 4 \Rightarrow F_4[x] = \{0, 1, x, x + 1\}$

Binary extension fields

A binary extension field is a set of polynomials

$$F_q[x] = \left\{ \sum_{i=0}^{n-1} a_i x^i \;\middle|\; a_i \in \mathbb{F}_2 \right\}$$

of order $q = 2^n$ with appropriately defined binary operators $\langle +, \cdot \rangle$.

Examples:

- $q = 2 \;\Rightarrow\; F_2[x] = \{0, 1\}$ (same as $\mathbb{F}_2$)
- $q = 4 \;\Rightarrow\; F_4[x] = \{0, 1, x, x + 1\}$

Addition of any $a, b \in F_q[x]$ is defined as

$$a(x) + b(x) = \sum_{i=0}^{n-1} a_i x^i + \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{n-1} (a_i + b_i) x^i,$$

where $+$ means XOR, which makes sense considering that the polynomials' coefficients are elements of $\mathbb{F}_2$.

Notational stuff

Since prime fields $\mathbb{F}_p$ are only a special case of binary extension fields $F_q[x]$, namely for $q = p^n = p$, we consider any finite field as (binary) extension field and denote

- the set of coefficients by $\mathbb{F}_p$ and
- the set of elements by $F_q[x]$.

If not noted otherwise, $p = 2$ is assumed.

What about multiplication?

- Multiplication of two polynomials of degree n and m yield another polynomial of degree at most $n + m$.
- Ordinary multiplication of $a, c \in F_q[x]$ would in general give a result $\notin F_q[x]$, violating the axiom of closure and therefore preventing us from calling it a finite (extension) field.

What about multiplication?

- Multiplication of two polynomials of degree n and m yield another polynomial of degree at most $n + m$.
- Ordinary multiplication of $a, c \in F_q[x]$ would in general give a result $\notin F_q[x]$, violating the axiom of closure and therefore preventing us from calling it a finite (extension) field.

The trick: reduce the multiplication result subject to some prime element (compare $\mathbb{F}_p$).

Irreducible polynomial (reduction polynomial)
A polynomial of degree n over the binary field $\mathbb{F}_p$ is called irreducible if it cannot be represented as product of two polynomials $a, c \in F_q[x]$ of degree strictly less than n. Such a polynomial is guaranteed to exist but in general not unique.

Multiplication of any two $a, c \in F_q[x]$ is defined as

$$b(x) = (a(x) \cdot c(x)) \bmod r(x),$$

where $r(x)$ is irreducible.

Example: $F_4[x] = \{0, 1, x, x + 1\}$ and $r(x) = x^2 + x + 1$ is irreducible.

Homework: Is it the only irreducible polynomial in $F_4[x]$?

Binary extension fields

Example: $F_4[x]$

- $F_4[x] = \{0, 1, x, x+1\}$
- $r(x) = x^2 + x + 1$

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Example: $F_4[x]$

- $F_4[x] = \{0, 1, x, x+1\}$
- $r(x) = x^2 + x + 1$

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

- Except for the highlighted values, the tables should be clear.
- Here is an example for the reduction of $x \cdot x$:

$$(x \cdot x) \bmod (x^2 + x + 1) = \begin{array}{l} x^2 \qquad\qquad \bmod (x^2 + x + 1) \\ -(x^2 + x + 1) \\ \hline -x - 1 = x + 1 \end{array}$$

Homework:

- Do the same for $F_8[x]$
- Does it work for $F_6[x]$?

Primitive element (generator)

Primitive element

A primitive element $g \in F_q[x]$ is a polynomial such that

$$\bigcup_{i=1}^{q-1} \left\{ g^i \right\} = F_q[x] \backslash \{0\},$$

and it is in general not unique.

Primitive elements are also referred to as generators. Each finite field has at least one. This also holds for residual classes $\mathbb{F}_p$.

Examples:

- For $\mathbb{F}_7$, $g = 3$ is a generator.
- For $F_4[x]$, $g(x) = x + 1$ is a generator.

Discrete logarithm

Primitive elements give raise for another multiplication algorithm: let

- $L[a]$ denote the discrete logarithm of $a \in F_q[x]$ and
- $A[a]$ the discrete power (antilog) of a.

Then, product and quotient of $a, b \in F_q[x]$ are given as

$$a \cdot b = A[L[a] + L[b]] \text{ and } a/b = A[L[a] - L[b]].$$

Discrete logarithm

Primitive elements give raise for another multiplication algorithm: let

- $L[a]$ denote the discrete logarithm of $a \in F_q[x]$ and
- $A[a]$ the discrete power (antilog) of a.

Then, product and quotient of $a, b \in F_q[x]$ are given as

$$a \cdot b = A[L[a] + L[b]] \text{ and } a/b = A[L[a] - L[b]].$$

First, we create two tables:

1. A containing all powers of g, i. e., $A[i] = g^i$
2. L containing the inverse elements, i. e., $L[g^i] = i$

For these steps we need to choose a reduction polynomial r. All calculations are done subject to this polynomial.

Multiplication $a \cdot b \bmod r$ is done as follows:

1. Determine $L[a]$ and $L[b]$, which yields the powers i, j such that $g^i = a$ and $g^j = b$.
2. Find $A[i + j]$, where addition is done modulo q (no XOR here).

Example: $F_{256}[x]$, $r(x) = x^8 + x^4 + x^3 + x + 1$, $g(x) = x + 1$

$$\underbrace{\left(x^2 + x\right)}\ \underbrace{\left(x^6 + x^4 + x + 1\right)} = ?$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	00	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
10	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
11	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
12	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
13	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
14	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
15	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

(a) L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
10	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
11	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
12	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
13	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
14	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
15	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

(b) A

Discrete logarithm

Example: $F_{256}[x]$, $r(x) = x^8 + x^4 + x^3 + x + 1$, $g(x) = x + 1$

$$\underbrace{\left(x^2 + x\right)}_{\texttt{0x06}} \underbrace{\left(x^6 + x^4 + x + 1\right)}_{} = ?$$

1. Lookup `0x06` and `0x53` in L, which yields `0x1a` and `0x30`, respectively.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	00	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
10	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
11	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
12	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
13	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
14	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
15	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

(c) L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
10	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
11	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
12	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
13	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
14	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
15	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

(d) A

Example: $F_{256}[x]$, $r(x) = x^8 + x^4 + x^3 + x + 1$, $g(x) = x + 1$

$$\underbrace{\left(x^2 + x\right)}_{\texttt{0x06}} \underbrace{\left(x^6 + x^4 + x + 1\right)}_{\texttt{0x53}} = ?$$

1. Lookup `0x06` and `0x53` in L, which yields `0x1a` and `0x30`, respectively.
2. Lookup `0x1a` + `0x30` mod `0xff` = `0x4a` in A.
3. This gives `0xf1` = $x^7 + x^6 + x^5 + x^4 + 1$, which is the result.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	00	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
10	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
11	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
12	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
13	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
14	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
15	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

(e) L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
10	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
11	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
12	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
13	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
14	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
15	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

(f) A

Discrete logarithm

Example: $F_{256}[x]$, $r(x) = x^8 + x^4 + x^3 + x + 1$, $g(x) = x + 1$

$$\underbrace{\left(x^2 + x\right)}_{\texttt{0x06}} \underbrace{\left(x^6 + x^4 + x + 1\right)}_{\texttt{0x53}} = ?$$

1. Lookup `0x06` and `0x53` in L, which yields `0x1a` and `0x30`, respectively.
2. Lookup `0x1a` + `0x30` mod `0xff` = `0x4a` in A.
3. This gives `0xf1` = $x^7 + x^6 + x^5 + x^4 + 1$, which is the result.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	00	00	19	01	32	02	1a	c6	4b	c7	1b	68	33	ee	df	03
1	64	04	e0	0e	34	8d	81	ef	4c	71	08	c8	f8	69	1c	c1
2	7d	c2	1d	b5	f9	b9	27	6a	4d	e4	a6	72	9a	c9	09	78
3	65	2f	8a	05	21	0f	e1	24	12	f0	82	45	35	93	da	8e
4	96	8f	db	bd	36	d0	ce	94	13	5c	d2	f1	40	46	83	38
5	66	dd	fd	30	bf	06	8b	62	b3	25	e2	98	22	88	91	10
6	7e	6e	48	c3	a3	b6	1e	42	3a	6b	28	54	fa	85	3d	ba
7	2b	79	0a	15	9b	9f	5e	ca	4e	d4	ac	e5	f3	73	a7	57
8	af	58	a8	50	f4	ea	d6	74	4f	ae	e9	d5	e7	e6	ad	e8
9	2c	d7	75	7a	eb	16	0b	f5	59	cb	5f	b0	9c	a9	51	a0
10	7f	0c	f6	6f	17	c4	49	ec	d8	43	1f	2d	a4	76	7b	b7
11	cc	bb	3e	5a	fb	60	b1	86	3b	52	a1	6c	aa	55	29	9d
12	97	b2	87	90	61	be	dc	fc	bc	95	cf	cd	37	3f	5b	d1
13	53	39	84	3c	41	a2	6d	47	14	2a	9e	5d	56	f2	d3	ab
14	44	11	92	d9	23	20	2e	89	b4	7c	b8	26	77	99	e3	a5
15	67	4a	ed	de	c5	31	fe	18	0d	63	8c	80	c0	f7	70	07

(g) L

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	01	03	05	0f	11	33	55	ff	1a	2e	72	96	a1	f8	13	35
1	5f	e1	38	48	d8	73	95	a4	f7	02	06	0a	1e	22	66	aa
2	e5	34	5c	e4	37	59	eb	26	6a	be	d9	70	90	ab	e6	31
3	53	f5	04	0c	14	3c	44	cc	4f	d1	68	b8	d3	6e	b2	cd
4	4c	d4	67	a9	e0	3b	4d	d7	62	a6	f1	08	18	28	78	88
5	83	9e	b9	d0	6b	bd	dc	7f	81	98	b3	ce	49	db	76	9a
6	b5	c4	57	f9	10	30	50	f0	0b	1d	27	69	bb	d6	61	a3
7	fe	19	2b	7d	87	92	ad	ec	2f	71	93	ae	e9	20	60	a0
8	fb	16	3a	4e	d2	6d	b7	c2	5d	e7	32	56	fa	15	3f	41
9	c3	5e	e2	3d	47	c9	40	c0	5b	ed	2c	74	9c	bf	da	75
10	9f	ba	d5	64	ac	ef	2a	7e	82	9d	bc	df	7a	8e	89	80
11	9b	b6	c1	58	e8	23	65	af	ea	25	6f	b1	c8	43	c5	54
12	fc	1f	21	63	a5	f4	07	09	1b	2d	77	99	b0	cb	46	ca
13	45	cf	4a	de	79	8b	86	91	a8	e3	3e	42	c6	51	f3	0e
14	12	36	5a	ee	29	7b	8d	8c	8f	8a	85	94	a7	f2	0d	17
15	39	4b	dd	7c	84	97	a2	fd	1c	24	6c	b4	c7	52	f6	01

(h) A

Discrete logarithm

So far we know about at least two different implementations of $\cdot$ over $F_q[x]$:

- Enumerate all results in a table of size q^2 (due to commutativity, a table of size $q^2/2$ is sufficient). That is possible since q is finite, but rather obvious and eventually inefficient.
- Do the log-table approach described before, which scales nicely with the field order q.

For small fields, both approaches are affordable and do not require support for any special instructions by the CPU.

Can we do better?

So far we know about at least two different implementations of $\cdot$ over $F_q[x]$:

- Enumerate all results in a table of size q^2 (due to commutativity, a table of size $q^2/2$ is sufficient). That is possible since q is finite, but rather obvious and eventually inefficient.
- Do the log-table approach described before, which scales nicely with the field order q.

For small fields, both approaches are affordable and do not require support for any special instructions by the CPU.

Can we do better? Yes, much better. Using vector instructions of today's CPUs allow for a variety of algorithms:

- simply do multiple operations at once, subject to limited register size and available instructions
- pipeline multiple operations, which exploits large registers
- break down the log-table approach to smaller scales, essentially doing $\log^2$ in register level
- heterogeneous multi processing, i. e., integrated GPUs sharing the virtual address space with the CPU may yield a significant advantage
- For GF(256), Intel introduced Galois Field New Instructions (GFNI) with Ice Lake resulting in up to $2\times$ performance increase compared to the fastest implementation using AVX512

$\Rightarrow$ speedup in orders of magnitude, coding operations are no longer a limiting factor for end devices – but what about network devices?

Linear coding operations

Encoding

- Data words are represented by elements $a \in F_q[x]$ of some binary extension field.
- A data packet is a sequence of M words and thus a vector $\boldsymbol{a} \in F_q^M[x]$.
- A generation of N packets is written as matrix $\boldsymbol{A} = [\boldsymbol{a}_1 \boldsymbol{a}_2 \ldots \boldsymbol{a}_N]$.
- A coded packet is obtained by

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{c} = \sum_{i=1}^{N} c_i \boldsymbol{a}_i,$$

  where $\boldsymbol{c} = [c_1, \ldots, c_N]^T \in F_q^N[x]$ denotes a vector of coding coefficients.

As a node generates multiple such linear combinations, we denote the k-th coded packet by $\boldsymbol{b}_k = \boldsymbol{A}\boldsymbol{c}_k$.

Note: On the whiteboard we have a hard time to differentiate vector $\boldsymbol{a}$ from scalar a and matrix $\boldsymbol{A}$ from set A. We therefore follow the notation I learned from Prof. Nossek and write

- $\underline{a}$ for vector $\boldsymbol{a}$ and
- $\underset{\sim}{A}$ for matrix $\boldsymbol{A}$

in class.

Transmission and recoding

- Packets are sent in general along with their coding vectors, i. e., $\boldsymbol{x}_k = \begin{bmatrix} \boldsymbol{c}_k \\ \boldsymbol{b}_k \end{bmatrix}$.
- Packets received by some node so far can therefore be written as $\boldsymbol{X}_k = [\boldsymbol{x}_1 \boldsymbol{x}_2 \ldots \boldsymbol{x}_k]$.
- Intermediate nodes may recode packets that have previously been received. Assume that some node has $k \leq N$ packets buffered, then

$$\boldsymbol{x}' = \sum_{i=1}^{k} c_i' \boldsymbol{x}_i = \boldsymbol{X}_k \boldsymbol{c}'$$

  is the recoded packet, where $c_i' \in F_q[x]$ are coding coefficients chosen by the intermediate node.
- Note that the original coding vector $\boldsymbol{c}$ is also recoded, i. e., the same linear transformation is applied to both the payload and the coding vector.

Decoding

- The receiver buffers coded packets as matrix $\boldsymbol{X}_k$:

$$\boldsymbol{X}_k = \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 & \dots & \boldsymbol{c}_k \\ \boldsymbol{b}_1 & \dots & \boldsymbol{b}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_k \\ \boldsymbol{B}_k \end{bmatrix}$$

- Noting that $\boldsymbol{B}_k = \boldsymbol{A}\boldsymbol{C}_k$, we obtain:

$$\begin{bmatrix} \boldsymbol{C}_k \\ \boldsymbol{B}_k \end{bmatrix} = \begin{bmatrix} \mathbf{1}\boldsymbol{C}_k \\ \boldsymbol{A}\boldsymbol{C}_k \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \boldsymbol{A} \end{bmatrix} \boldsymbol{C}_k$$

- For $k = N$ we have $\boldsymbol{C}_N \in F_q^{N \times N}[x]$.
- If the columns of $\boldsymbol{C}_N$ are linear independent, than rank $\boldsymbol{C}_N = N$ and $\boldsymbol{C}_N^{-1}$ such that $\boldsymbol{C}_N \boldsymbol{C}_N^{-1} = \mathbf{1}$ exists.
- $\boldsymbol{C}_N^{-1}$ can be determined using gaussian elimination.[4]
- Decoded packets are indicated by the columns of the unit matrix $\mathbf{1}$.
- Decoded packets are naturally in-order.

$\Rightarrow$ The destination can decode a generation after receiveing N linear independent coded packets.

[4] who not remembers gaussian elemination should look it up until next time.

Observations:

- The coding matrix $\boldsymbol{C}$ fully describes the state of a generation:
  - rank $\boldsymbol{C}$ denotes the number of (possibly coded) packets in that generation.
  - $\boldsymbol{C} = \mathbf{1}$ means that all packets are decoded.
  - If $\boldsymbol{C}$ is in column-echelon[5] form, then a subset of packets may be decoded.
  - Recoding operations applied to $\boldsymbol{C}$ (or to a subset of its columns) are correspondingly applied to the packets (or a subset of those).
- The coding matrix at each node spans a vector space span $\boldsymbol{C}$, which is a subspace of the N-dimensional vector space $F_q^N[x]$.
- The dimension dim $\boldsymbol{C}$ and its changes over time are a concise description of the state of the network.

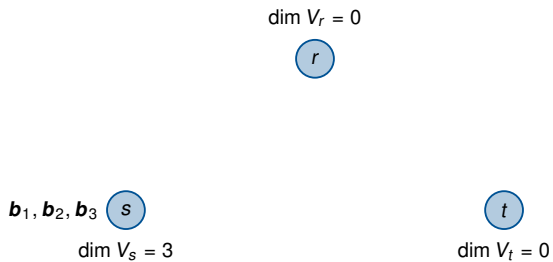
Note: To solve a given $\boldsymbol{X}$ by hand, it may be useful to consider $\boldsymbol{X}^T$ instead:

- Rows in $\boldsymbol{X}^T = [\boldsymbol{C}^T \boldsymbol{B}^T]$ represent coded packets
- Solving $\boldsymbol{C}^T$ is probably more intuitive as operations are applied to rows instead of columns

[5] more on that later

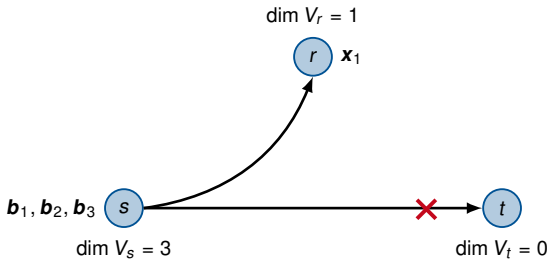
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have $\dim V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



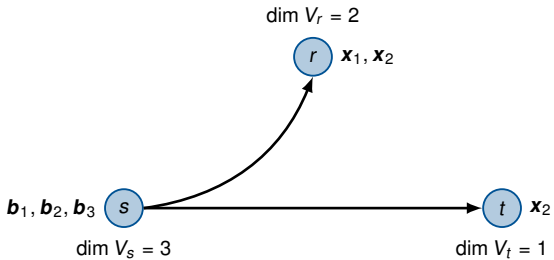
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have $\dim V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



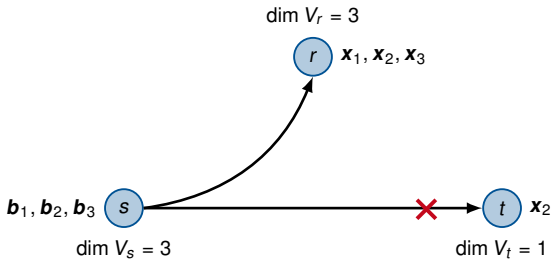
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have dim $V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



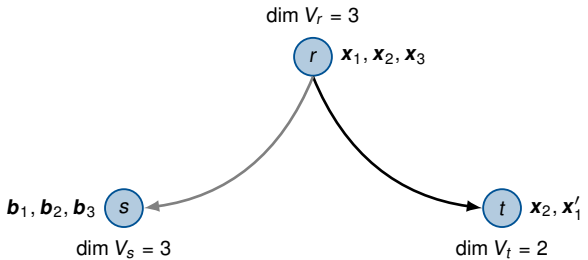
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have $\dim V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



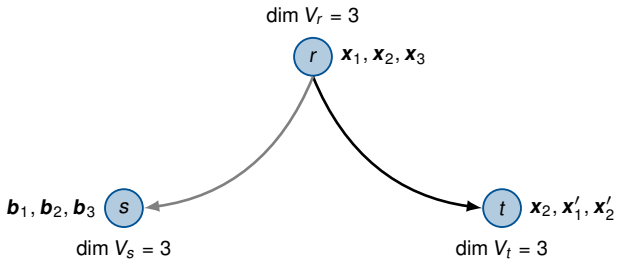
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have dim $V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



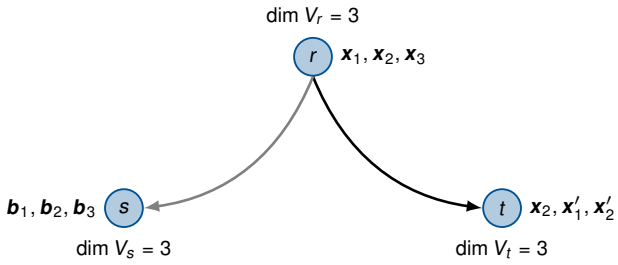
Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have dim $V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



Example

- Erasure network with symmetric loss probabilities ϵ_{st}, ϵ_{sr}, and ϵ_{rt}.
- Unidirectional traffic from s to t.
- We assume that randomly choosen coding vectors are all linear independent.
- V_i is the subspace known to node i, in particular we have dim $V_s = N$ and $V_r, V_t \subset V_s$.
- To keep it short we assume $N = 3$.



The recoded packets $\boldsymbol{x}'_j$ for $j \in \{1, 2\}$ can be written as $\boldsymbol{x}'_j = \boldsymbol{X}\boldsymbol{c}'_j = \sum_{i=1}^{3} c'_{ji}\boldsymbol{x}_i$.

Flows and sessions

Definition: (unicast) flow
In the context of network coding, we define an unicast flow (s, t) as the sequence of packets originating at some source node s and destined for precisely one destination node t.

Definition: (unicast) session
In the context of network coding, we define an unicast session as the tuple $\langle (s, t), (t, s) \rangle$ of two (unicast) flows in opposite directions.

Above definitions naturally extend to (single source) multicasts.

Intra-session vs. inter-session coding

Intra-session coding:

- Only packets belonging to the same session may be coded together.
- Flows belonging to the same session may be coded together provided that bidirectional coding is allowed.
- Easier to implement, but fewer coding opportunities and thus potentially lower coding gain.

Inter-session coding:

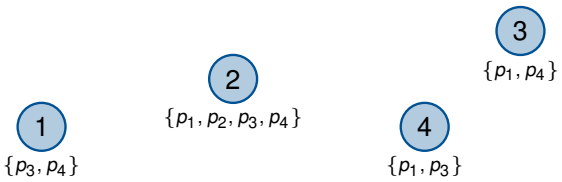
- Packets of arbitrary flows / sessions may be combined.
- More coding opportunities but also more complex.

We will mainly focus on intra-session coding (particular in projects).

COPE – XORs in the air [5]

Example:

- wireless nodes 1 – 4 with packet buffers denoted below each node
- node 2 is allowed to transmit a single packet that may be an arbitrary XOR of buffered packets
- no erasures and all nodes are in range of each other
- next hops: $p_1 \rightarrow 1$, $p_2 \rightarrow 3$, $p_3 \rightarrow 3$, $p_4 \rightarrow 4$

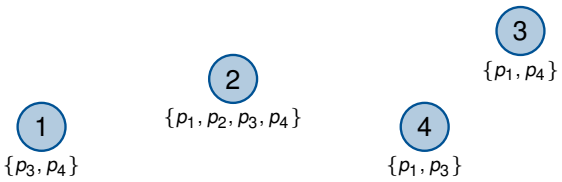


Question: which linear combination should node 2 transmit?

COPE – XORs in the air [5]

Example:

- wireless nodes 1 – 4 with packet buffers denoted below each node
- node 2 is allowed to transmit a single packet that may be an arbitrary XOR of buffered packets
- no erasures and all nodes are in range of each other
- next hops: $p_1 \rightarrow 1$, $p_2 \rightarrow 3$, $p_3 \rightarrow 3$, $p_4 \rightarrow 4$



Question: which linear combination should node 2 transmit?

1. $p_1 \oplus p_2$ would allow node 3 to decode p_2 but no other node receives information it is intersted in
2. $p_1 \oplus p_3$ allows both node 1 and 3 to decode
3. $p_1 \oplus p_3 \oplus p_4$ allows all receivers to decode

COPE – XORs in the air [5]

COPE is an inter-session coding scheme that

- exploits (and requires) broadcast media and
- assures decodability at each hop.

Its basic principle is to transmit linear combinations of packets such that

1. the number of neighbors receiving new information is maximized while
2. assuring that all neigbors are able to decode their respective packets.

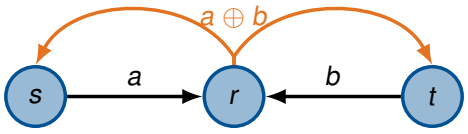
Problem: For optimal decisions a transmitter has to know the exact buffer contents of all its neighbors, which is impossible in general:

- Transmissions may be overheard by any subset of neighbors.
- Acknowledgements (“reception reports”) from neighbors are also subject to erasures.
- When overhearing a transmission it is impossible to tell which other nodes were also able to overhear.

$\Rightarrow$ COPE estimates the neighbor states based on monitored link qualities.

Relationship to bidirectional intra-session coding

- In gernal, COPE allows packets from arbitrary sessions to be coded together (s. t. decodability constraints), thus inter-session.
- In a three-node relay network it becomes similar (but not identical) to bidirectional intra-session coding:



COPE:

- s, t send only linear combinations that allow r to decode.
- r sends in turn only linear combinations that allow both s, t to decode.

Random linear intra-session coding:

- There is no need for r to decode anything.
- Transmissions of r are random linear combinations of both directions (flows).

MORE – a MAC-independent opportunistic routing protocol [1]

- Originally proposed by Chachulski [1, 2] in 2007
- Unidirectional intra-session network coding
- Coding done between layer 2 and 3
- Opportunistic routing

MORE – a MAC-independent opportunistic routing protocol [1]

- Originally proposed by Chachulski [1, 2] in 2007
- Unidirectional intra-session network coding
- Coding done between layer 2 and 3
- Opportunistic routing

MORE framing:

MORE – a MAC-independent opportunistic routing protocol [1]

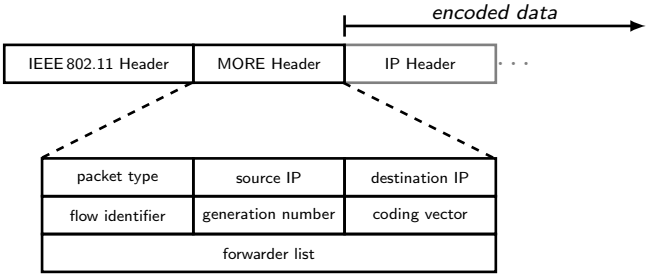
- Originally proposed by Chachulski [1, 2] in 2007
- Unidirectional intra-session network coding
- Coding done between layer 2 and 3
- Opportunistic routing

MORE framing:

To achieve opportunistic routing, MORE has

- to send packets as MAC-layer broadcasts or
- use IEEE 802.11 MAC in promiscuous or monitor mode.

MORE header

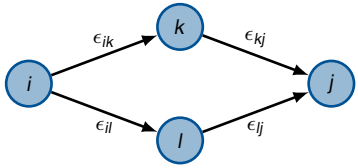


- packet type differentiates between data packets and acknowledgements
- flow identifier is used to identify the flow coded packets belong to
- generation number identifies a specific generation of packets within a flow
- coding vector is the vector $\boldsymbol{c} \in F_q^N[x]$ of random coefficients
- forwarder fields are used for routing ← more on that later

Routing metric used by MORE

MORE uses the ETX (estimated transmission count) [3] metric:

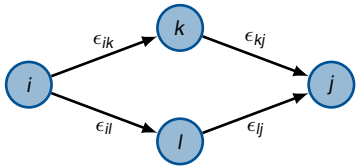
- The distance d_{ij} between two nodes i, j is the estimated number of transmissions along the best path between i, j.
- Consider the network below, assuming that there is no link between k and l:



Routing metric used by MORE

MORE uses the ETX (estimated transmission count) [3] metric:

- The distance d_{ij} between two nodes i, j is the estimated number of transmissions along the best path between i, j.
- Consider the network below, assuming that there is no link between k and l:



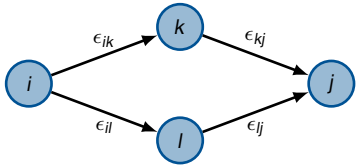
- Then the ETX distance d_{ij} in this example is

$$d_{ij} = \min \left\{ \frac{1}{1-\epsilon_{ik}} + \frac{1}{1-\epsilon_{kj}}, \frac{1}{1-\epsilon_{il}} + \frac{1}{1-\epsilon_{lj}} \right\}$$

Routing metric used by MORE

MORE uses the ETX (estimated transmission count) [3] metric:

- The distance d_{ij} between two nodes i, j is the estimated number of transmissions along the best path between i, j.
- Consider the network below, assuming that there is no link between k and l:



- Then the ETX distance d_{ij} in this example is

$$d_{ij} = \min\left\{\frac{1}{1-\epsilon_{ik}} + \frac{1}{1-\epsilon_{kj}}, \frac{1}{1-\epsilon_{il}} + \frac{1}{1-\epsilon_{lj}}\right\}$$

Note: $d_{ik} = \dfrac{1}{1-\epsilon_{ik}}$ is the expectation of a random variable $X \sim \text{Geo}(1-\epsilon_{ik})$.

MORE – a MAC-independent opportunistic routing protocol [1]

MORE uses opportunistic routing, i. e.,

- nodes overhear all transmissions and
- decide *somehow* whether or not to recode.

But *how*?

- Nodes estimate erasure probabilities by sending beacons.
- From this information the ETX distance between any $i \in \mathcal{N}$ and destination t can be computed.
- Let z_i the average number of transmissions by some $i \in \mathcal{N}$ per source packet.
- We denote nodes $i \in \mathcal{N}$ with higher ETX distance than some other note $j \in \mathcal{N}$ by $i > j$.
- The expected number of packets that j receives from nodes with higher ETX distance per source packet is given as

$$R_j = \sum_{i>j} z_i \left(1 - \epsilon_{ij}\right).$$

Expected number of packets j has to forward for each source packet sent by s:

- To avoid redundant transmissions node j should forward only if no other node k with lower distance to t has received a specific packet.
- The expected number of packets that node j has to send is therefore

$$L_j = \sum_{i>j} \left(z_i \left(1 - \epsilon_{ij}\right) \prod_{k<j} \epsilon_{ik} \right).$$

- Note that $L_s = 1$.

Expected number of packets j has to forward for each source packet sent by s:

- To avoid redundant transmissions node j should forward only if no other node k with lower distance to t has received a specific packet.
- The expected number of packets that node j has to send is therefore

$$L_j = \sum_{i>j} \left(z_i \left(1 - \epsilon_{ij}\right) \prod_{k<j} \epsilon_{ik} \right).$$

- Note that $L_s = 1$.

Expected number of transmissions at node j per source packet:

- j has to transmit encoded packets until L_j packets have been received by nodes with lower ETX distance to t.
- That is given by

$$z_j = \frac{L_j}{1 - \prod_{k<j} \epsilon_{jk}}.$$

TX credit counter

- In order to keep track of how many packets nodes may send, each $i \in \mathcal{N}$ maintains a TX credit counter Z_i^{TX}.
- Z_i^{TX} is decremented by 1 for every packet that is transmitted.
- Node i stops transmitting if $Z_i^{\mathrm{TX}} < 0$ (note that $Z_i^{\mathrm{TX}} \in \mathbb{R}$).

TX credit counter

- In order to keep track of how many packets nodes may send, each $i \in \mathcal{N}$ maintains a TX credit counter Z_i^{TX}.
- Z_i^{TX} is decremented by 1 for every packet that is transmitted.
- Node i stops transmitting if $Z_i^{\mathrm{TX}} < 0$ (note that $Z_i^{\mathrm{TX}} \in \mathbb{R}$).

Problem

- Each node has calculated its value z_i denoting the number of packets i has to send for each packet generated by s.
- However, i cannot know how many packets s generates.

TX credit counter

- In order to keep track of how many packets nodes may send, each $i \in \mathcal{N}$ maintains a TX credit counter Z_i^{TX}.
- Z_i^{TX} is decremented by 1 for every packet that is transmitted.
- Node i stops transmitting if $Z_i^{\mathrm{TX}} < 0$ (note that $Z_i^{\mathrm{TX}} \in \mathbb{R}$).

Problem

- Each node has calculated its value z_i denoting the number of packets i has to send for each packet generated by s.
- However, i cannot know how many packets s generates.

Solution

- In practice, Z_i^{TX} should be incremented whenever i receives a packet from some node with higher ETX distance.
- This incremental update is given by

$$\Delta Z_i^{\mathrm{TX}} = \frac{z_i}{R_i} = \frac{z_i}{\sum_{j>i} z_j \left(1 - \epsilon_{ji}\right)}.$$

Which nodes should act as forwarders?

With MORE, the subgraph that participates in forwarding packets from s to t is determined by s:

- Each packet sent by s contains a list of nodes that are closer to t than s ordered by their distance.
- Whenever an intermediate node overhears a coded packet, it first checks this forwarder list and discards the packet if it is not included in the list of possible forwarders.

The use of such a forwarder list

- resembles a variant of source routing and
- allows to dynamically setup routes instead of calculating all possible routes in advance,
- which limits the overhead induced by routing updates.

Acknowledgements (aka feedback)

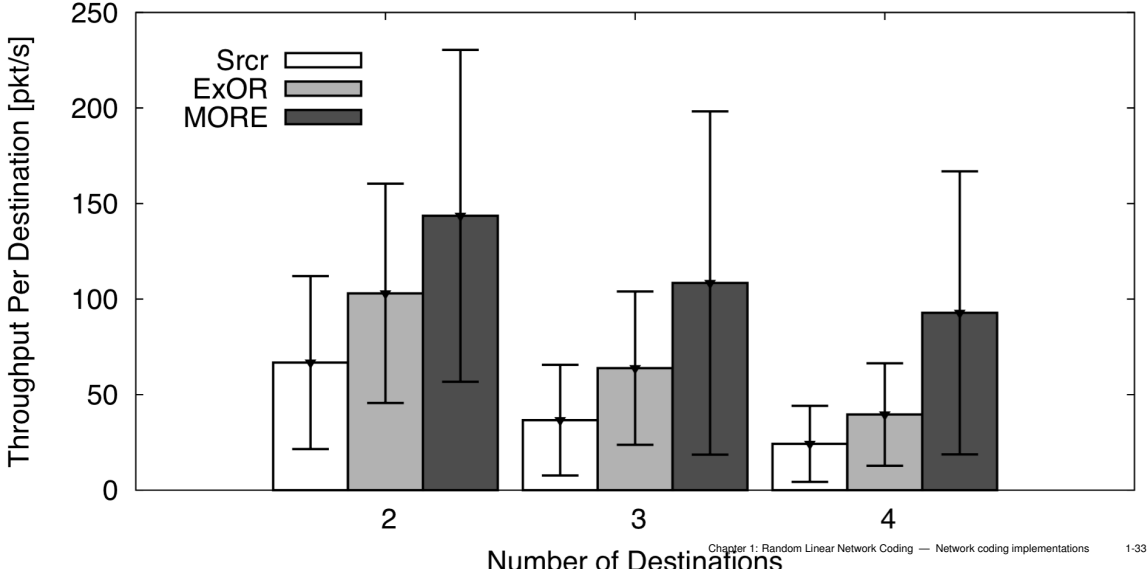
Source s keeps transmitting redundant frames until successful decoding is acknowledged by the destination t:

- t sends an uncoded acknowledgement back to s containing the generation sequence number.
- The acknowledgement is routed along the shortest path from t to s according to the ETX metric.
  - The MORE acknowledgement is treated as IEEE 802.11 data frame and
  - thus acknowledged on the link layer
  - leading to a near-zero loss probability of acknowledgements.
- Forwarders overhearing the acknowledgement prune the corresponding generation.

But at which rate should s keep transmitting?

- The expected number of transmissions needed is given by z_s
- However, that is only an expectation and in general not sufficient
- Simply transmitting as fast as possible would lead to a lot of superfluous redundant packets . . .

Throughput gain of MORE [2]



What is FEC?

- FEC systematically adds redundancy to a transmission (at the symbol / packet / file / block-level depending on layer and application).
- It allows to detect and correct certain classes of transmission errors.
- We assume for now erasures, i. e., we want to correct lost (missing) symbols or blocks.

The code rate R is a measure of how much redundancy is added. Assuming an FEC code that maps k source symbols to $n \geq k$ output symbols, the code rate is defined as $R = k/n$.

We can differentiate between fixed-rate and rateless codes.

- Fixed-rate codes have a predefined R that remains constant, e. g. Hamming codes, Reed-Solomon codes, Bose-Chaudhuri-Hocquenghem (BCH) codes.
- Rateless codes (fountain codes) are, in principle, able to output an endless sequence of coded symbols. R is not fixed but may vary over time for the very same code, e. g. Luby transform (LT) codes, Raptor codes.

Example: Luby transform codes

The transmitter divides a packet into N blocks of equal size and then encodes blocks as follows:

1. Randomly choose $1 \leq d \leq N$ blocks, where d is the degree of the encoded block.
2. XOR exactly these d blocks.
3. Transmit the encoded block along with its degree, the list of indices (positions of blocks used for encoding), and a checksum.

It is obvious that the receiver can decode after receiving a sufficient number of encoded blocks.

Example: Luby transform codes

The transmitter divides a packet into N blocks of equal size and then encodes blocks as follows:

1. Randomly choose $1 \leq d \leq N$ blocks, where d is the degree of the encoded block.
2. XOR exactly these d blocks.
3. Transmit the encoded block along with its degree, the list of indices (positions of blocks used for encoding), and a checksum.

It is obvious that the receiver can decode after receiving a sufficient number of encoded blocks.

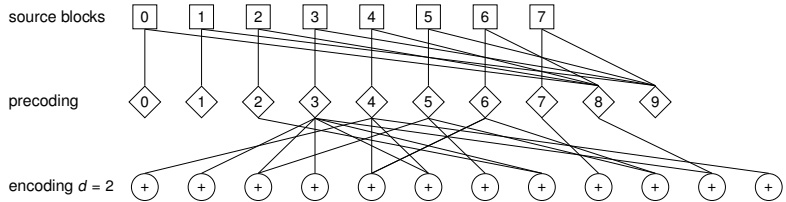
What's the difference to network coding?

- Coding operations are performed at the source node only. Alternatively, intermediate nodes may decode blocks and encode from scratch. With NC, intermediate nodes may re-encode packets without decoding.

Example: Raptor codes

Work similar to the LT codes but

1. use degree $d \ll N$ for combining blocks (inner code) and
2. employ a precoding stage (outer code) to recover blocks that are not transmitted due to sparsity.



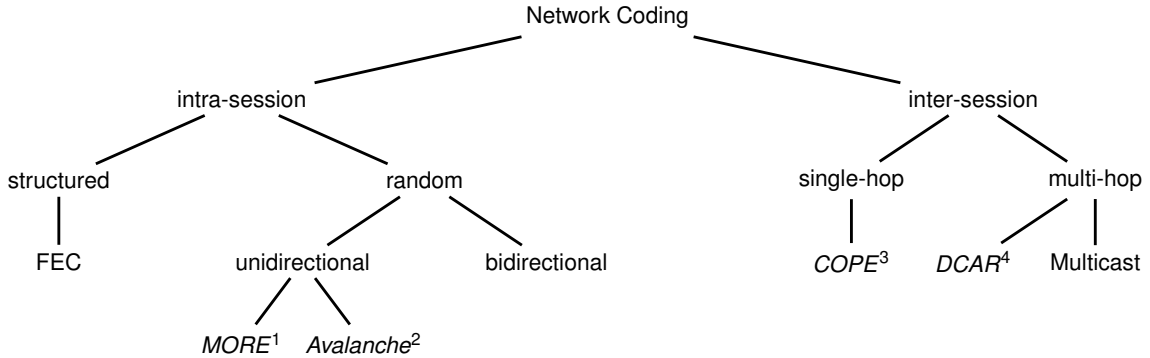
Raptor codes are thus

- a concatenation of two codes (inner and outer), where
- the inner code is some kind of LT code.

Raptor codes vs. random linear network codes

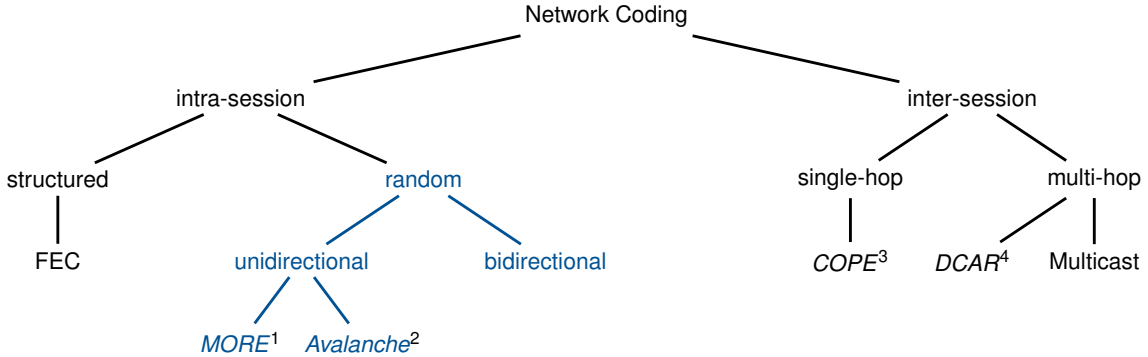
- Encoding/Decoding
  - Raptor codes have a particular sparsity structure that enables fast encoding and decoding operations.
  - Network codes have no particular structure: decoding via Gauss elimination.
- Recoding at intermediate nodes
  - Raptor structure is lost when partially recoded at intermediate nodes. The structure could only be conserved if they knew all source data.
  - Network codes can be arbitrarily recoded from any set of coded blocks.

A possible classification of network coding:



[1] *Trading Structure for Randomness in Wireless Opportunistic Routing*, Chachulski, S. [1]
[2] *Avalanche: File Swarming with Network Coding*, Gkantsidis, C. and Goldberg, M. [4]
[3] *XORs in the Air: Prcatical Wireless Network Coding*, Katti, S. et al. [5]
[4] *DCAR: Distributed Coding-Aware Routing in Wireless Networks*, Le, J. et al [6]

A possible classification of network coding:



The lecture mostly covers random linear intra-session network coding.

[1] *Trading Structure for Randomness in Wireless Opportunistic Routing*, Chachulski, S. [1]
[2] *Avalanche: File Swarming with Network Coding*, Gkantsidis, C. and Goldberg, M. [4]
[3] *XORs in the Air: Prcatical Wireless Network Coding*, Katti, S. et al. [5]
[4] *DCAR: Distributed Coding-Aware Routing in Wireless Networks*, Le, J. et al [6]

Bibliography

[1] S. Chachulski.
Trading Structure for Randomness in Wireless Opportunistic Routing.
M.sc. thesis, Massachusetts Institute of Technology, 2007.

[2] S. Chachulski, M. Jennings, S. Katti, and D. Katabi.
Trading Structure for Randomness in Wireless Opportunistic Routing.
In *ACM SIGCOMM*, pages 169–180, 2007.

[3] D. De Couto, D. Aguayo, J. Bicket, and R. Morris.
A High-throughput Path Metric for Multi-hop Wireless Routing.
In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, MobiCom 2003, pages 134–146, New York, NY, USA, 2003. ACM.

[4] C. Gkantsidis and M. Goldberg.
Avalanche: File Swarming with Network Coding.
http://research.microsoft.com/en-us/projects/avalanche/.

[5] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft.
XORs in the Air: Practical Wireless Network Coding.
16(3):497–510, Jun. 2008.

[6] J. Le, J. Lui, and D.-M. Chiu.
DCAR: Distributed Coding-Aware Routing in Wireless Networks.
9(4):596–608, Apr. 2010.