Reduction of Decoding Iterations for Zigzag Decodable Fountain Codes

Takayuki Nozaki

Yamaguchi University
This work is partially supported by JSPS KAKENHI Grant Number 16K16007

ISITA2016
Nov. 2nd, 2016

Outline

Fountain code

Rateless error correcting code for user diagram protocol (UDP)
(Applications) Multicasting, Broadcasting
(Examples) LT code, Raptor code, RaptorQ code ...

Previous work [Nozaki2014]

Proposing "Zigzag decodable (ZD) fountain code"
(Strength) Small overhead, Low decoding erasure rate
(Weakness) Large number of decoding iterations/time

Purpose of this research

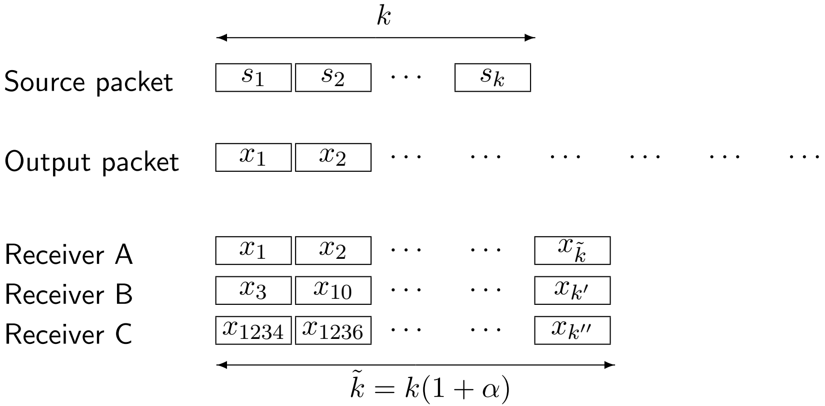
Reduction of number of decoding iterations for ZD fountain code
(Result) Reducing decoding iterations without loss of decoding performance

Outline

1. Fountain code and Raptor code
2. Encoding/Decoding for ZD fountain code
3. Investigation of number of iterations for conventional decoding algorithm
4. Proposed decoding algorithm
5. Simulation results

Fountain code (1: Brief review)

[Encoder] generates many output packets from k source packets.
[Decoder] decodes source packets from *any* $k(1+\alpha)$ received packets
$\Rightarrow$ Suitable for multicasting/broadcasting

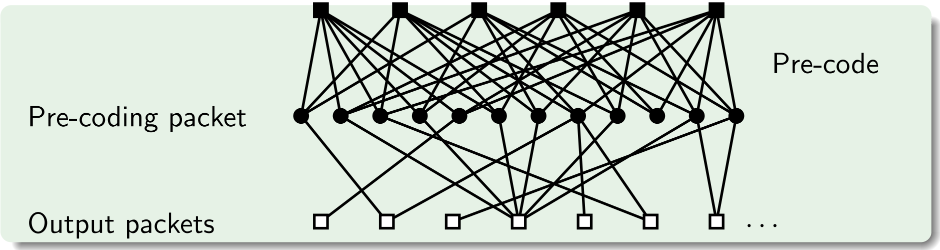


Even if packet losses occur, receiver can recover source packets

Fountain Code (2: Raptor code)

Raptor code $(\mathcal{C}, \Omega(x))$

- Encoding
  1. Generate pre-coding packets from source packets by using precode $\mathcal{C}$
  2. Generate output packets from pre-coding packets by using LT code $\Omega(x) = \sum_i \Omega_i x^i$
     1. Choose degree d with probability Ω_d
     2. Choose distinct d pre-coding packets $(\boldsymbol{s}_{j_1}, \boldsymbol{s}_{j_2}, \ldots \boldsymbol{s}_{j_d})$
     3. The output packet is $\sum_{i=1}^{d} \boldsymbol{s}_{j_i}$
- Decoding
  - Generate factor graph from received packets and $\mathcal{C}$
  - Execute peeling algorithm (PA)



ZD Fountain Code (1: Toy Example)

Encoding

Using XOR and bit-level shift for source packets

Source packet	$s_{1,1}$	$s_{1,2}$	$\cdots$	$s_{1,\ell}$	
	$s_{2,1}$	$s_{2,2}$	$\cdots$	$s_{2,\ell}$	
Output packet	$x_{1,1}$	$x_{1,2}$	$\cdots$	$x_{1,\ell}$	

$s_{1,1}$	$s_{1,2}$	$\cdots$	$s_{1,\ell}$	
	$s_{2,1}$	$\cdots$	$s_{2,\ell-1}$	$s_{2,\ell}$
$x_{2,1}$	$x_{2,2}$	$\cdots$	$x_{2,\ell}$	$x_{2,\ell+1}$

Decoding: Bit-wise peeling algorithm (Zigzag decoding)

- Recover source packets in bit-wise

1. $s_{1,1} = x_{2,1}$
2. $s_{2,1} = x_{1,1} - s_{1,1}$
3. $s_{1,2} = x_{2,2} - s_{2,1}$

Zigzag Decodable Fountain Code (2: Encoding)

ZD Fountain Code $(\mathcal{C}, \Omega(x), \Delta(x))$

1. Generate pre-coding packets $(\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_n)$ from source packets $(\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_k)$ by using precode $\mathcal{C}$
2. Generate an output packet as follows:
   1. Choose degree d of an output packet with probability Ω_d
   2. Choose d distinct precoded packets. Denote those indexes of packets by $(j_1, j_2, \dots, j_d)$.
   3. Choose d-tuple of shift amount $(\delta_1, \delta_2, \dots, \delta_d)$ according to $\Delta(x) = \sum_{i=0}^{D} \Delta_i x^i$
   4. Send the following output packet

$$\sum_{i=1}^{d} z^{\delta_i} a_{j_i}(z),$$

Polynomial representation of a packet

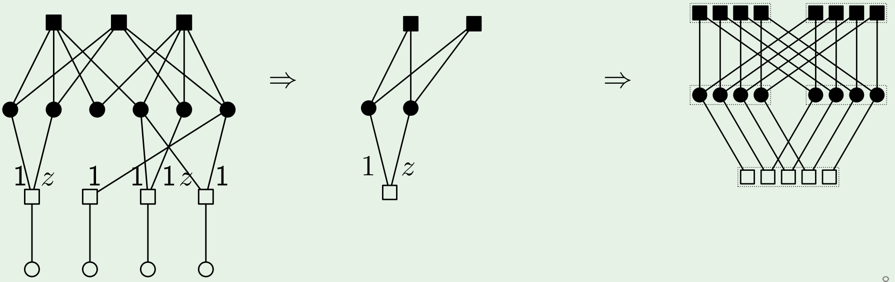
$$a_j(z) = a_{j,1} + a_{j,2} z + a_{j,3} z^3 + \cdots + a_{j,\ell} z^{\ell-1}$$

$$\begin{aligned} a_1(z) + z a_2(z) = \ & a_{1,1} + a_{1,2} z + a_{1,3} z^3 + \cdots + a_{1,\ell} z^{\ell-1} \\ & + a_{2,1} z + a_{2,2} z^3 + \cdots + a_{2,\ell-1} z^{\ell-1} + a_{2,\ell} z^{\ell} \end{aligned}$$

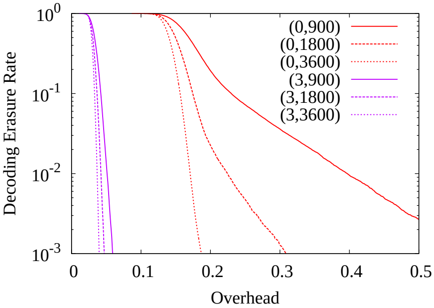
Zigzag Decodable Fountain Code (3: Decoding)

Conventional Decoding Algorithm

1. Construct factor graph in packet-wise representation from precode $\mathcal{C}$ and received packets $(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{k'})$
2. (Packet-wise PA) Peeling algorithm over the factor graph
3. IF residual graph is empty, decoding succeeds and halts. Otherwise go to next step.
4. Transform the residual graph into bit-wise representation
5. (Bit-wise PA) Peeling algorithm over the bit-wise factor graph



Performance Comparison (Raptor vs ZD Fountain)



Decoding time [sec] $(\alpha = 0.12)$

	Raptor	ZD Fountain
$\ell = 100$	0.17039	0.43909
$\ell = 1000$	0.17039	14.0874

Red: Raptor code,
Purple: ZD Fountain code
$k = 900, 1800, 3600$

Strength and Weakness of ZD fountain code

Strength: Small overhead and small decoding erasure rate
Weakness: Large decoding time

Goal and Strategy of Research

(Goal of Research): reduction of number of decoding iterations without loss of decoding performance

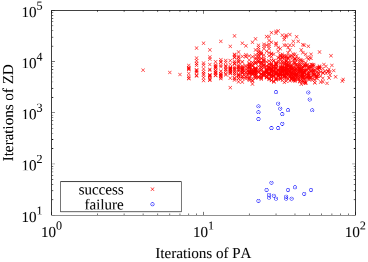
(From simulation result)

Decoding iterations are mainly caused by bit-wise decoding

$\Rightarrow$ The source packets are mainly recovered by bit-wise decoding

Strategy of Research: Using a powerful decoding algorithm for packet-wise decoding.

(Decreasing the iterations of bit-wise decoding)



The number of iterations under pPA+bPA ($\ell = 1000$, $\alpha = 0.07$)

Decoding Algorithms for LDPC Codes over BEC

Decoding algorithms works upon factor graphs

PA (Peeling Algorithm) [Luby et al. 1997]

- Decoding starts from check nodes of degree 1
- Low complexity, Low decoding performance

TEP (Tree-structure Expectation Propagation) [Olmos et al. 2010]

- Decoding starts from check nodes of degree 1 and 2
- Moderate complexity, Moderate decoding performance

G-TEP (Generalized TEP) [Salamanca et al. 2013]

- Decoding starts from check nodes of any degree
- Equivalent to MAP decoding (i.e, Gaussian elimination)
- High complexity, High decoding performance

Proposed Algorithm

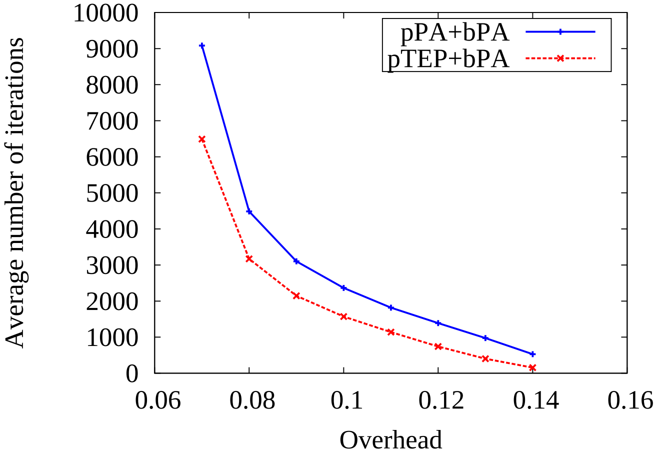
Applying TEP decoding algorithm for the packet-wise decoding of ZD fountain code (packet-wise TEP + bit-wise PA)

1. Initialize the residual graph $\mathtt{G}$ by the Tanner graph corresponding to $\mathbf{H}$. Initialize all the memory as $s_i \leftarrow 0$ $(i \in [1, m])$ and the iteration round τ as 1. For $j \in [1, n]$ s.t. $y_j \neq *$, update the memory in the check node $\mathtt{c}_i$ $(i \in \mathcal{N}_{\mathrm{v}}(j))$ as $s_i \leftarrow s_i + y_j$ and remove the j-th variable node and its connecting edges from $\mathtt{G}$. Additionally, set $F \leftarrow \{\}$.
2. For $i \in [1, m]$, execute the following processes;
   1. If the i-th check node is degree 1 in $\mathtt{G}$, execute the following For $i \in [1, m]$, if the i-th check node is degree 1 in $\mathtt{G}$, then the algorithm executes the following; Let j be the index of the adjacent variable node. The j-th variable node sets $b_j(z) \leftarrow \ell_{i,j}^{-1} s_i(z)$ and send $b_j(z)$ to all the adjacent check nodes. For $k \in \mathcal{N}_{\mathrm{v}}(j)$, the check node $\mathtt{c}_k$ updates the memory as $s_k(z) \leftarrow s_k(z) + \ell_{k,j} b_j(z)$. Remove the variable node $\mathtt{v}_j$ and its connecting edges from $\mathtt{G}$.
   2. If the i-th check node is degree 2 in $\mathtt{G}$, execute the followings. Let j, j' be the indexes of the adjacent variable nodes. Assume that the degree of the j-th variable node is less than or equal to that of j'-th variable node. For all $t \in \mathcal{N}_{\mathrm{v}}(j) \setminus \{i\}$, let g_t be the greatest common divisor of $l_{i.j}$ and $l_{t,j}$ and change the labels and memory as follows:

   $$\begin{aligned} &m_t(z) \leftarrow [l_{i,j} m_t(z) + l_{t,j} m_i(z)]/g_t, \\ &l_{t,j} \leftarrow 0, \\ &l_{t,\tilde{j}} \leftarrow l_{i,j} l_{t,\tilde{j}}/g_t \quad \forall \tilde{j} \in \mathcal{N}_{\mathrm{c}}(t) \setminus \{j\}, \\ &l_{t,j'} \leftarrow l_{t,j'} + l_{t,j} l_{i,j'}/g_t. \end{aligned}$$

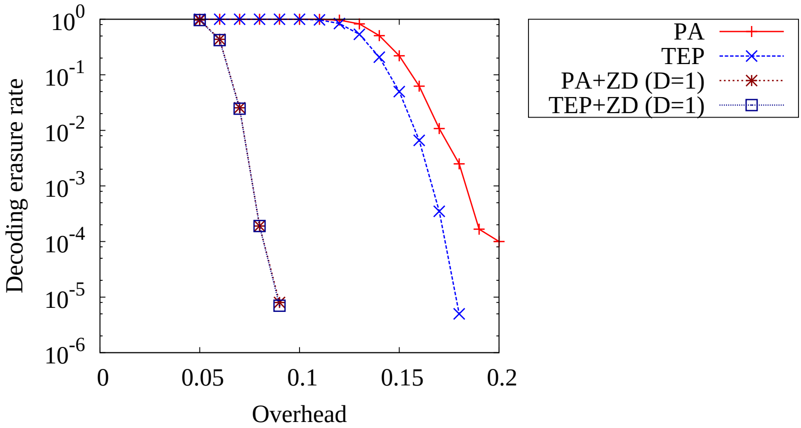
   Moreover, set $F \leftarrow F \cup \{i\}$.
   3. If there exist some check nodes of degree 1 or 2 in $\mathtt{G}$, set $\tau \leftarrow \tau + 1$ and go to Step 2. Otherwise, output the decoding result $b_1(z), \ldots, b_n(z)$, the residual graph $\mathtt{G}$ and memory values $s_1(z), \ldots, s_{m'}(z)$.

Simulation Results (1: Average Number of Iteration)



$\ell = 1000$

Simulation Results (2: Decoding Performance)



Decoding performance ($\ell = 1000$)

- There are no degradation of decoding performance.

Conclusion and Future Works

Conclusion

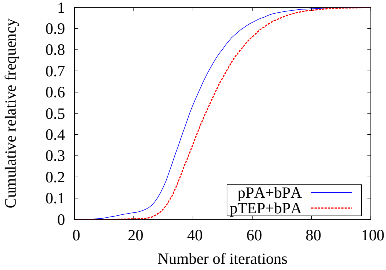
We propose a decoding algorithm for ZD fountain code

- Small number of iterations
- No degradation of decoding performance

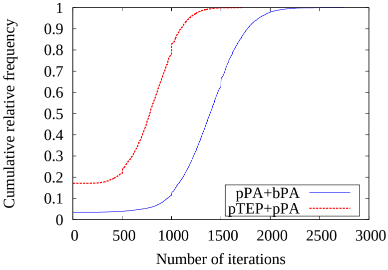
Future works

- Optimization of degree distribution $\Omega(x)$ and shift distribution $\Delta(x)$
- Comparison of MAP threshold and PA threshold for ZD fountain code

Simulation Results (3: Details of Decoding Iterations)



The number of iterations for packet-wise decoding



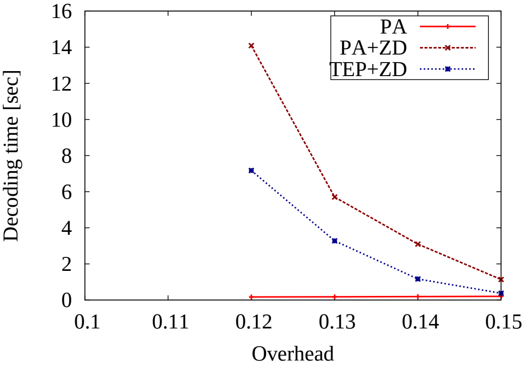
The number of iterations for bit-wise decoding

$\ell = 1000, \alpha = 0.12$

We reduce the total number of decoding iteration

- The number of packet-wise decoding is increasing (5 iterations)
- The number of bit-wise decoding is decreasing (600 iterations)

Simulation Results (4: Decoding Time)



Time of decoding ($\ell = 1000$)

- We can reduce the latency of decoding