Fault Tolerant Decoders

Predrag Ivaniš[1], **Bane Vasić**[2]

[1]*School of Electrical Engineering, University of Belgrade, Serbia*

[2]*Department of ECE, University of Arizona, Tucson, USA*

E-mail: [1]predrag.ivanis@etf.rs, [2]vasic@ece.arizona.edu

Abstract

In this talk we introduce a novel class of fault-tolerant decoders for low-density parity check codes, based on bit-flipping decoding algorithm. Presented decoding algorithm is not only superior to other decoding algorithms of this type, but also robust to logic gate failures.

Key words: Bit flipping, Fault tolerance, Iterative decoders, Low density parity check codes

Synopsis

According to new design paradigm for Very Large Scale Integration (VLSI) technologies, due to lower supply voltages and variations in technological process, fully reliable operations are not guaranteed in nano-scale devices [1]. A hardware component is assumed to be unreliable if it is subject to so-called transient faults, i.e. faults that manifest themselves at particular time instants but do not necessarily persist for later times [2]. An integral part of many such systems, designed for communications or computing, is error-control coding whose role is to maintain the data integrity. Thus, analysis of different decoding algorithms for low density parity check (LDPC) codes under unreliable hardware is meaningful. The density evolution analysis of sum-product algorithm (SPA) [3] and Gallager B algorithm [4] demonstrate robustness of these algorithms to the transient failures.

The reliable storage of data in a memory built from unreliable logic gates under transient failures can be achieved by employing LDPC codes and simple bit-flipping (BF) decoding [5]. In this talk we focus on modifications of BF algorithms that are suitable if only bit hard decisions are available. Although the performances of BF decoder are typically inferior when compared to the Gallager-B algorithm, we report the decoding algorithm based on BF with significant performance improvement, that has large immunity to the gate failures.

Fault tolerant decoders for BSC

We start by a overview of hard decision decoders based on BF and Gallager A/B algorithms. The structure of variable node processors will be explained and implementation of these decoders in unreliable hardware will be considered. Further, we will explain Gradient Descent Bit Flipping (GDBF) algorithm [6], where the inverse function is represented in the form that is more suitable for BSC. In this algorithm, the most critical value of the inverse function determines the bits that should be flipped in the current iteration. This algorithm is suitable for hardware implementation as it can be designed by using XOR and ML gates. Then, we propose version of GDBF decoder where the probabilistic mechanism is incorporated in the decoder structure by using generator of uniform random numbers. In this algorithm, the most critical value of the modified function represents only necessary condition for flipping.

We will show that the logic gate failures can improve performances of GDBF decoder, in contrary to classical BF decoders. By using the knowledge about trapping sets, we optimize the probabilistic mechanism to improve the performance of GDBF decoder realized in faulty hardware. It will be shown that the proposed solution outperforms Gallager-B algorithm and have performances compared to more complex massage passing algorithms. This is a hard-decision algorithm with the best known performances in the water-fall region on BSC, robust to the failures in logic gates.

Acknowledgment

This work was supported by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project). It is also funded in part by the Ministry of Education, Science and Technological Development of the Republic of Serbia under grant TR32028 and NSF under grants CCF-0963726 and CCF-1314147.

References

[1] S. Ghosh and K. Roy, “Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era,” *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.

[2] C. N. Hadjicostis and G. C. Verghese, “Coding Approaches to Fault Tolerance in Linear Dynamic Systems,” *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 210-228, Jan. 2005.

[3] L. Varshney, “Performance of LDPC Codes Under Faulty Iterative Decoding,” *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, July 2011.

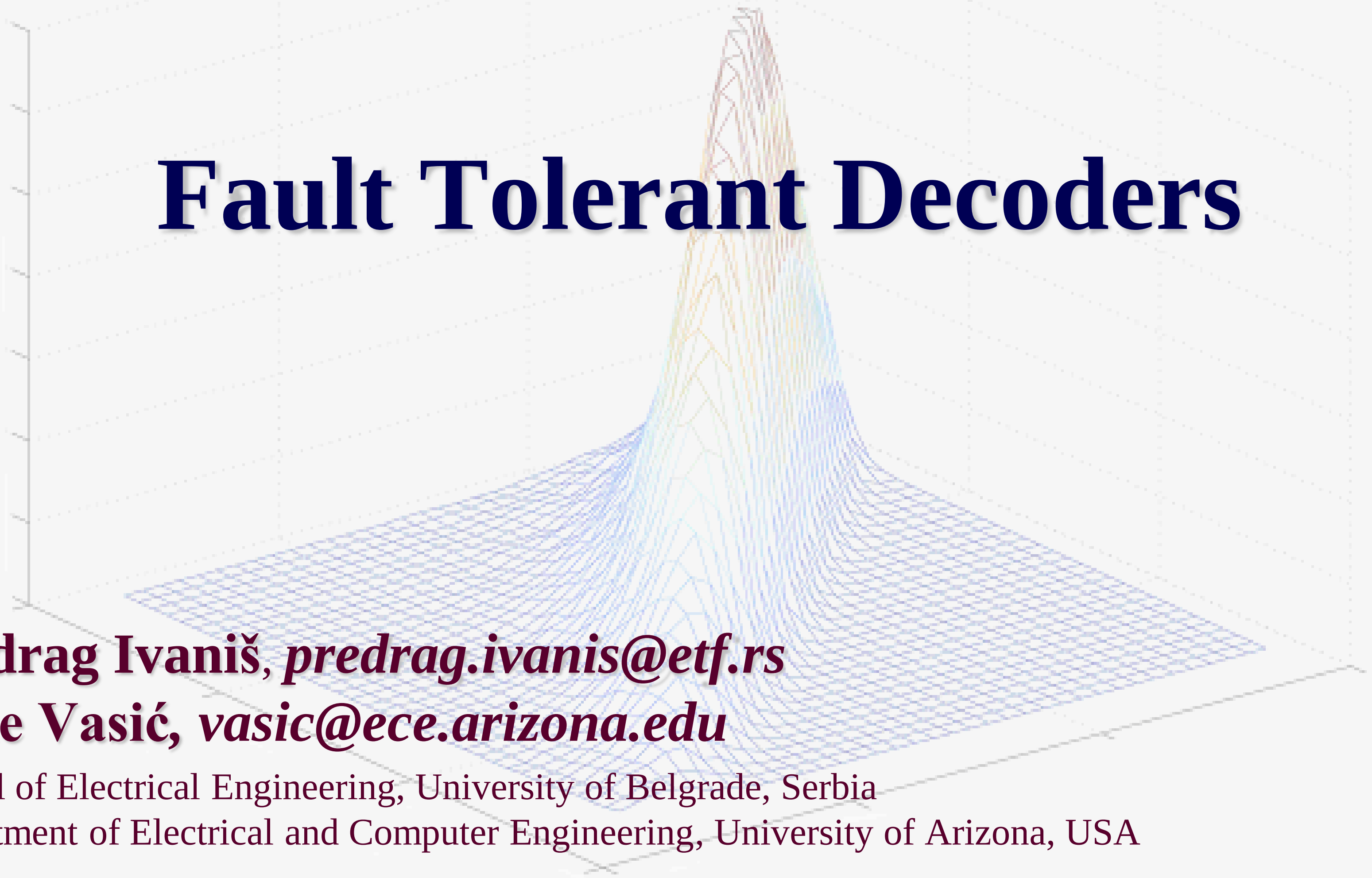
[4] S. Tabatabaei Yazdi, H. Cho, and L. Dolecek, “Gallager B Decoder on Noisy Hardware,” *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1660–1673, May 2013.

[5] B. Vasic and S. K. Chilappagari, “An information theoretical framework for analysis and design of nanoscale fault-tolerant memories based on low-density parity-check codes,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2438–2446, Nov. 2007.

[6] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, “Gradient Descent Bit Flipping algorithms for decoding LDPC codes,” *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614,

TINKOS 2014, Niš, June 2014

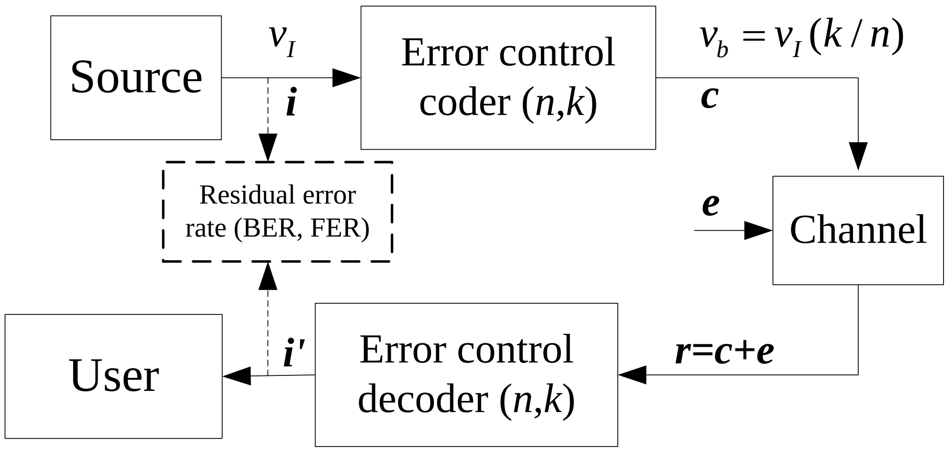
Fault Tolerant Decoders



Predrag Ivaniš, ***predrag.ivanis@etf.rs***
Bane Vasić, ***vasic@ece.arizona.edu***
School of Electrical Engineering, University of Belgrade, Serbia
Department of Electrical and Computer Engineering, University of Arizona, USA

Why faulty decoders?

- The Second Shannon theorem - the reliable transmission can be provided if the code rate *R*=*k*/*n* is smaller than the transmitted information *I*(*X*,*Y*) for a given channel.
- Increased integration factor of integrated circuits, stringent energy-efficiency constraints -> a new design paradigm for Very Large Scale Integration (VLSI) technology.
- Fully reliable operation of hardware components is not guaranteed!



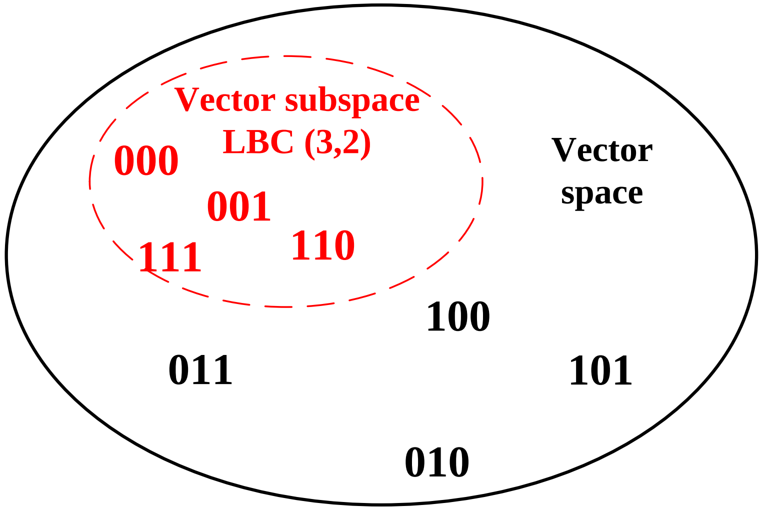
Source
v_I
$\boldsymbol{i}$
Error control coder (*n*,*k*)
$v_b = v_I(k/n)$
$\boldsymbol{c}$
Residual error rate (BER, FER)
$\boldsymbol{e}$
Channel
User
$\boldsymbol{i}'$
Error control decoder (*n*,*k*)
$\boldsymbol{r}=\boldsymbol{c}+\boldsymbol{e}$

Overview

- Linear block codes, LDPC codes
- Gallager-B algorithm
- Performance evaluation of QC-LDPC codes with Gallager-B decoding with transient failures
- Faulty bit-flipping algorithm
- Faulty gradient descent bit flipping (GDBF)
- Faulty Probabilistic GDBF

Linear block codes

- (3,2) code, field GF(2)



$$\mathbf{G} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}, \quad \boldsymbol{c} = \boldsymbol{i}\mathbf{G}.$$

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & g_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & g_{n-k,n} \end{bmatrix}, \quad \boldsymbol{S} = \boldsymbol{r}\mathbf{H}^{\mathrm{T}}.$$

- Vector space with 2^n ellements with length n=3.
- Vector subspace with 2^k elemens with length n=3. This vector subspace (set of codewords) is linear block code (3,2).

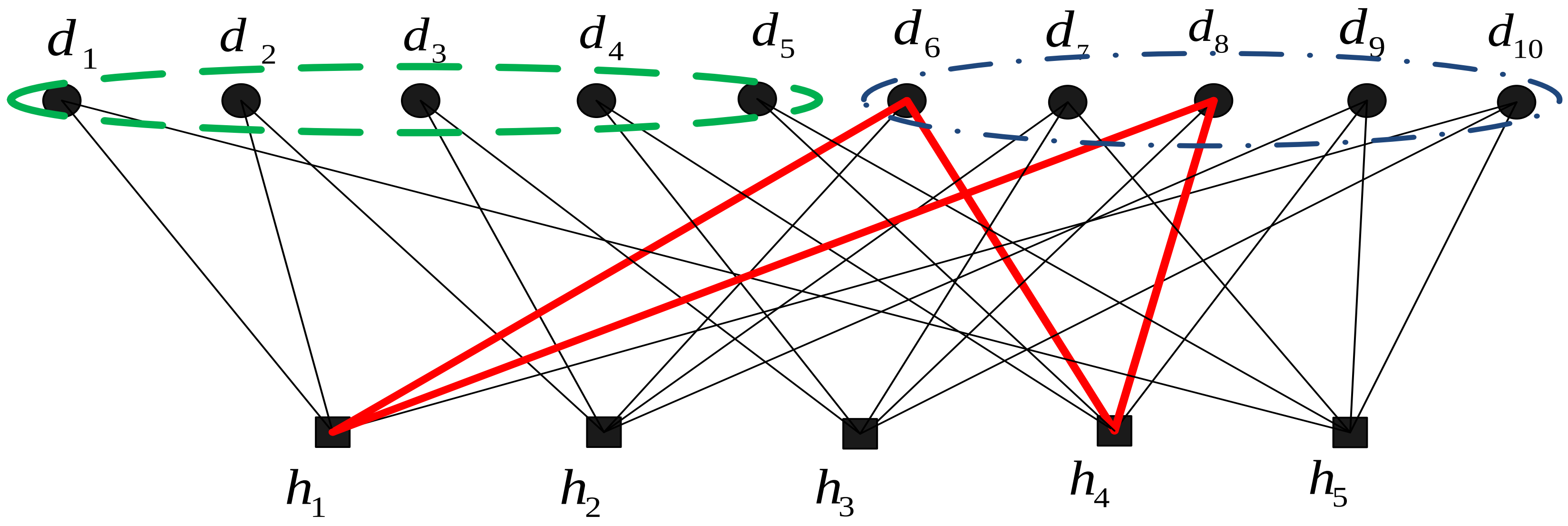
LDPC codes - properties

- Completely defined by parity check matrix **H**.
- Regular or irregular (fixed or variable number of ones per row/column)
- *Sparse* matrix (low density of binary one in matrix **H**)

$$\mathbf{H} = \left[\begin{array}{ccccccccccccccc}
 & 1 & & & & & & & & & & & & & \\
 & & & & & & & & & & & 1 & & & \\
 & & & 1 & & & & & & & & & & & \\
 & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & & & & & 1 \\
 & & & & 1 & & & & & & 1 & & & & \\
1 & & & & & & & & & & & & & & \\
 & & & & & & & & & 1 & & & & & \\
 & & 1 & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & &
\end{array}\right]$$

Tanner bipartite graph

- QC LDPC – suitable for hardware implementation



- Regular code with girth equal to 4!

Gallager-B, iterative decoding

- The simplest massage-passing algorithm

Initialization (i=1): For each variable node v, and each set $E(v)$, messages sent to check nodes are computed as follows

$$m_1(e) = r(v). \tag{2}$$

Step (i) (*check-node update*): For each parity check node c and each set $E(c)$, update rule for i-th iteration, $i > 1$, is defined as follows

$$m_i'(e) = \left(\sum_{e' \in E(c) \setminus \{e\}} m_{i-1}(e) \right) \bmod 2. \tag{3}$$

Step (ii) (*variable-node update*): For each variable node v and each set $E(v)$, update rule for i-th iteration, $i > 1$, is defined as follows

$$m_i(e) = \begin{cases} 1, & if \quad \sum_{e' \in E(v) \setminus \{e\}} m_i'(e) \geq \lceil d_v / 2 \rceil \\ 0, & if \quad \sum_{e' \in E(v) \setminus \{e\}} m_i'(e) \leq d_v - 1 - \lceil d_v / 2 \rceil, \\ r(v), & otherwise. \end{cases} \tag{4}$$

An example

- Regular Euclidean geometry LDPC code with:
  - ρ=4 (weight of check node - number of ones in every row),
  - γ=4 (weight of variable node - number of ones in every column)

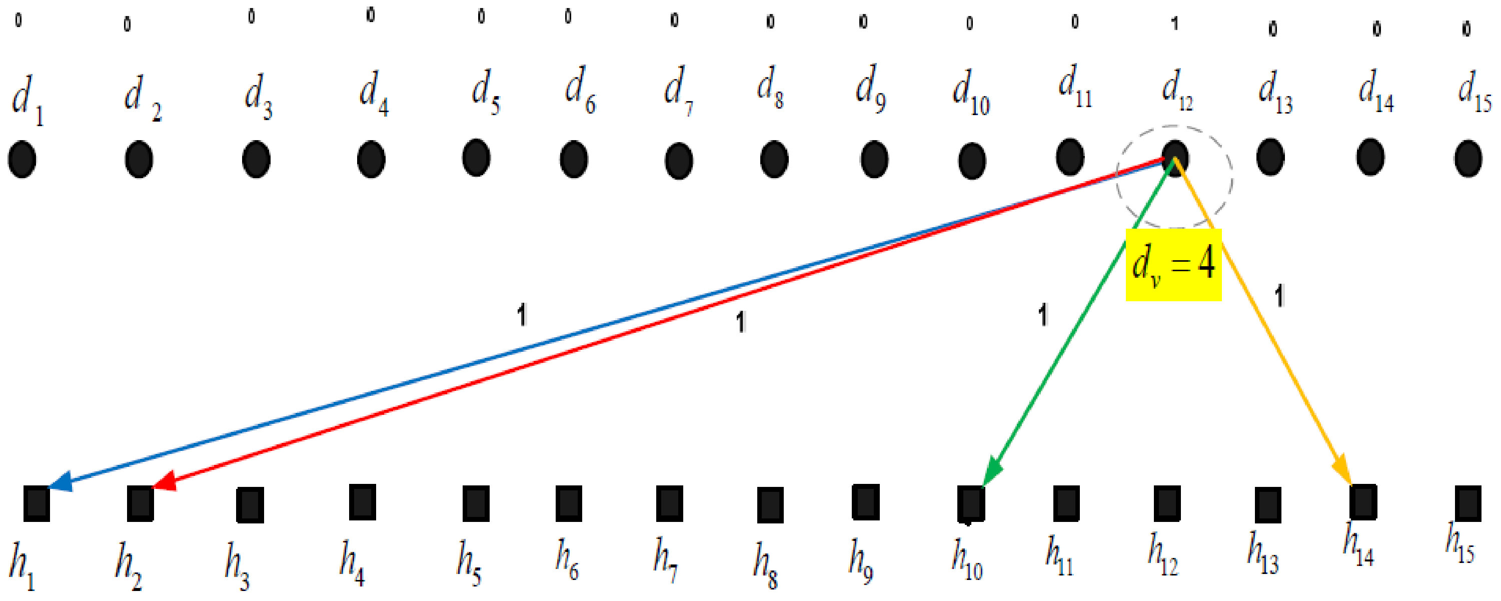
$$
H = \begin{array}{ccccccccccccccc|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 2 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 3 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 4 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 8 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 10 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 13 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 14 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 15
\end{array}
$$

Gallager-B, initialization

- For variable node d_{12} incident edges are d_{12}->h_1, d_{12}->h_2, d_{12}->h_{10} and d_{12}->h_{14} and messages that are initially sent are:

$$m_1(d_{12}\text{->}h_1)= m_1(d_{12}\text{->}h_2)= m_1(d_{12}\text{->}h_{10})= m_1(d_{12}\text{->}h_{14})=r(12)=1,$$

and the messages sent over all other edges are equal to binary zero (as the other bits in received vector are equal to zero).

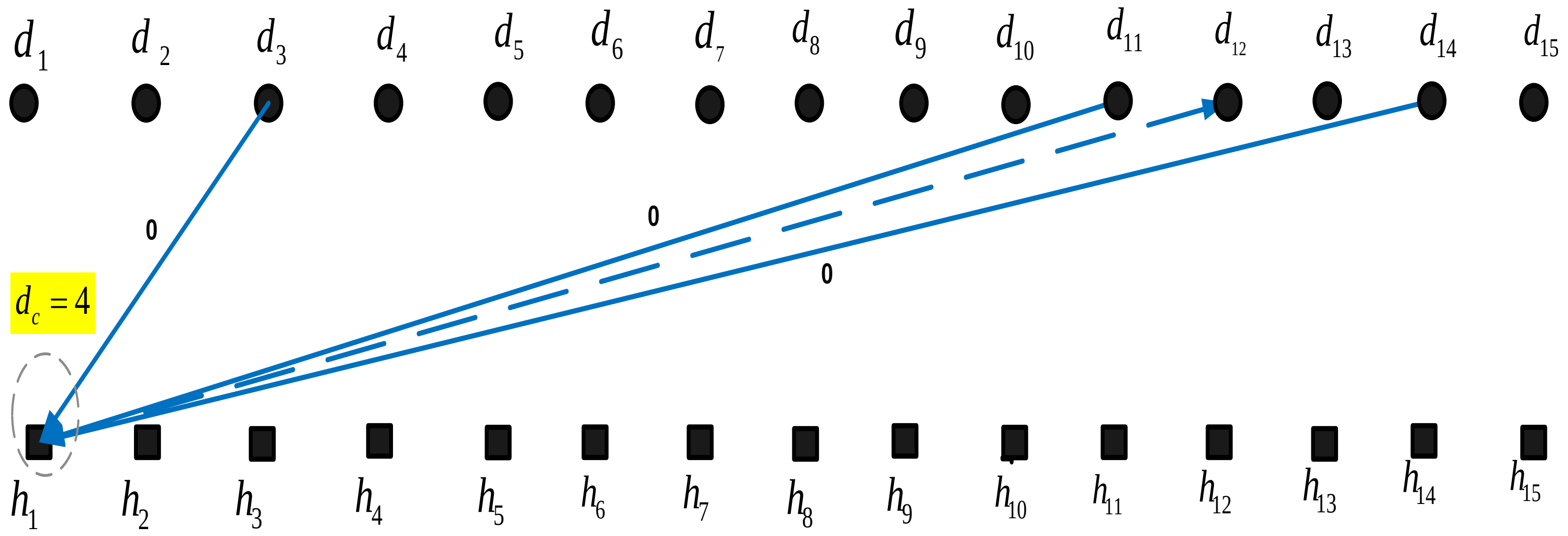


0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15}
$d_v=4$
1
1
1
1
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11} h_{12} h_{13} h_{14} h_{15}

Gallager-B, *step (i)- check-node update*

- Update for check node 1:

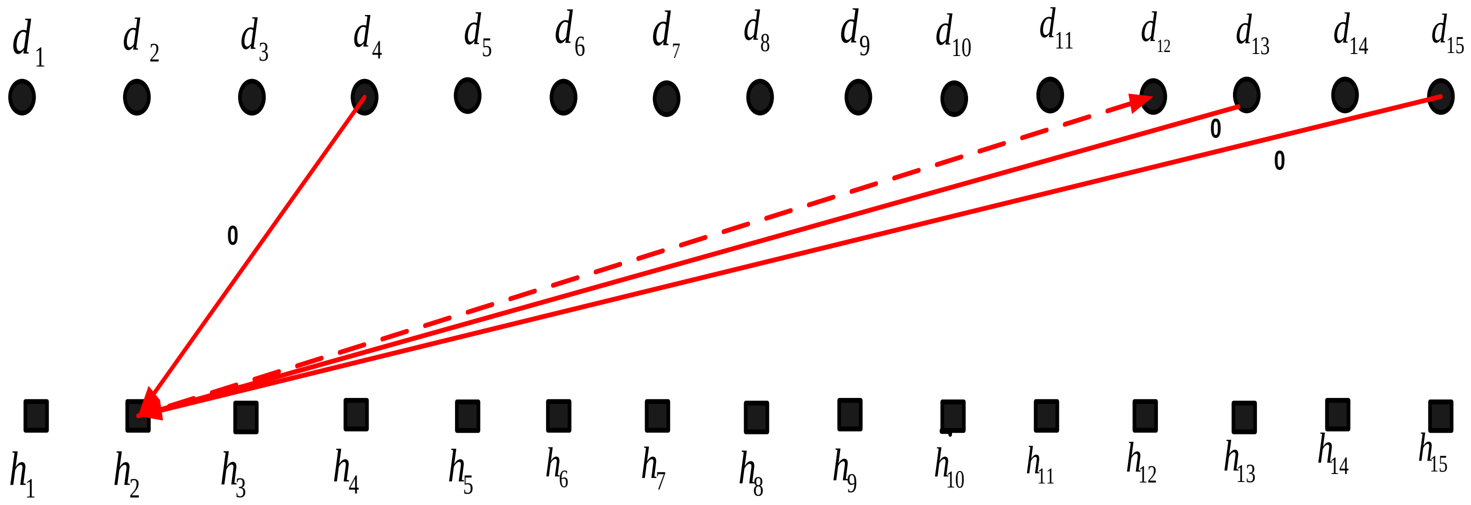
$$m_2'(h_1 -> d_{12}) = \left(m_1(d_3 -> h_1) + m_1(d_{11} -> h_1) + m_1(d_{14} -> h_1)\right) \bmod 2$$
$$= \left(0+0+0\right) \bmod 2 = 0.$$


d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15}
0
0
0
$d_c = 4$
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11} h_{12} h_{13} h_{14} h_{15}

Gallager-B,
step (i)- check-node update

- Update for check node 2:

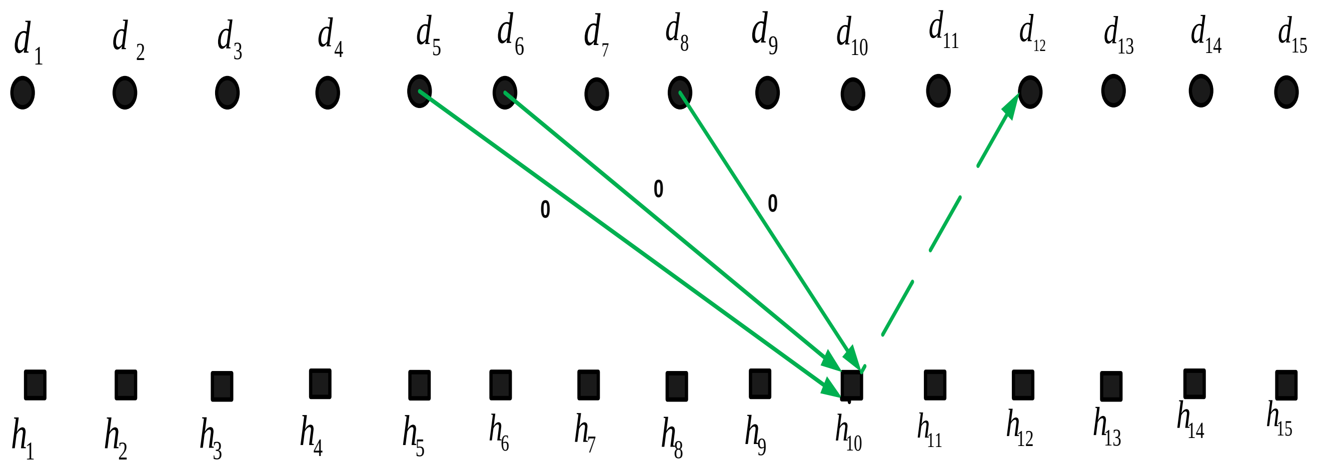
$$m_2'(h_2 -> d_{12}) = \left(m_1(d_4 -> h_2) + m_1(d_{13} -> h_1) + m_1(d_{15} -> h_1)\right) \bmod 2$$
$$= (0+0+0) \bmod 2 = 0.$$



Gallager-B, *step (i)- check-node update*

- Update for check node 10:

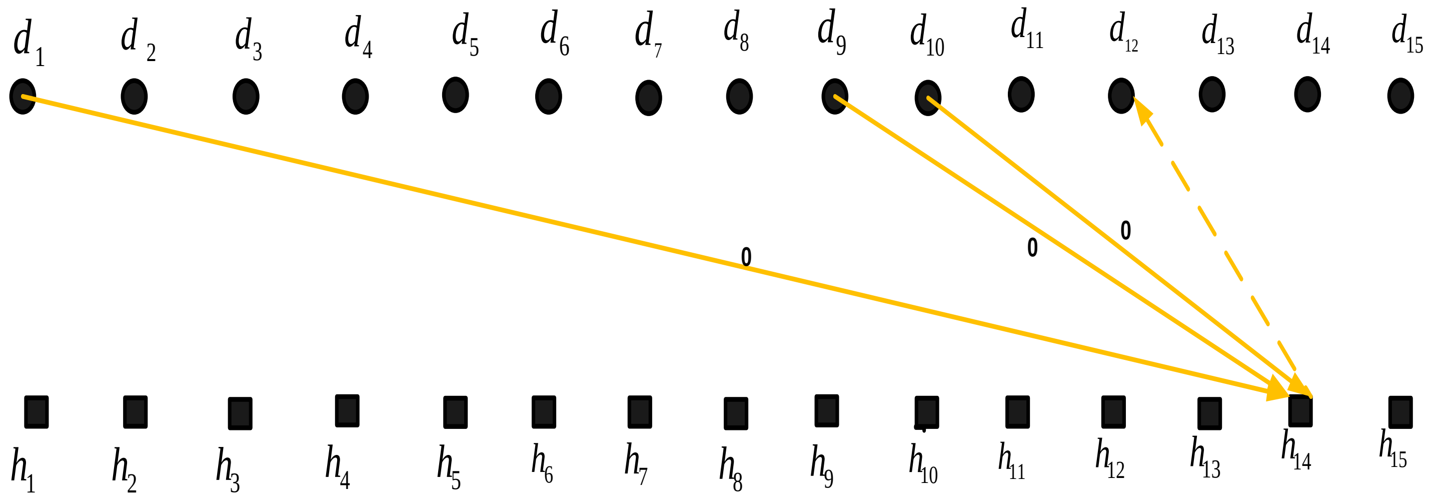
$$m_2'(h_{10}->d_{12})=\left(m_1(d_5->h_{10})+m_1(d_6->h_{10})+m_1(d_8->h_{10})\right)\bmod 2$$
$$=\left(0+0+0\right)\bmod 2=0.$$


d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15}
0
0
0
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11} h_{12} h_{13} h_{14} h_{15}

Gallager-B, *step (i)- check-node update*

- Update for check node 14:

$$m_2'(h_{14}->d_{12})=\left(m_1(d_1->h_{14})+m_1(d_9->h_{14})+m_1(d_{10}->h_{14})\right)\bmod 2$$
$$=\left(0+0+0\right)\bmod 2=0.$$



d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15}
0
0
0
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11} h_{12} h_{13} h_{14} h_{15}

Gallager-B, *step (ii)- variable-node update*

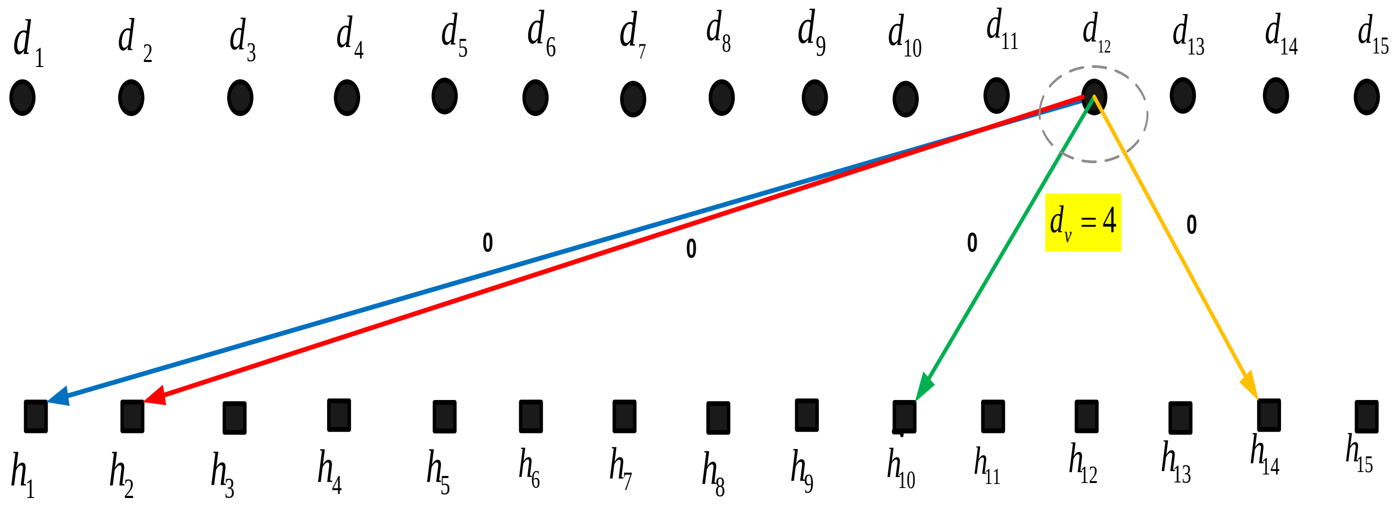
For variable node v=12 and set E(12)={ h_1->d_{12}, h_2->d_{12}, h_{10}->d_{12} and h_{14}->d_{12}} we obtain

$$\sum_{e'\in E(12)\backslash\{d_{12}->h_1\}} m_2'(e') = m_2'(h_2->d_{12}) + m_2'(h_{10}->d_{12}) + m_2'(h_{14}->d_{12}) = 0 \;\Rightarrow\; m_2(d_{12}->h_1) = 0$$

$$\sum_{e'\in E(12)\backslash\{d_{12}->h_2\}} m_2'(e') = m_2'(h_1->d_{12}) + m_2'(h_{10}->d_{12}) + m_2'(h_{14}->d_{12}) = 0 \;\Rightarrow\; m_2(d_{12}->h_2) = 0$$

$$\sum_{e'\in E(12)\backslash\{d_{12}->h_{10}\}} m_2'(e') = m_2'(h_1->d_{12}) + m_2'(h_2->d_{12}) + m_2'(h_{14}->d_{12}) = 0 \;\Rightarrow\; m_2(d_{12}->h_{10}) = 0$$

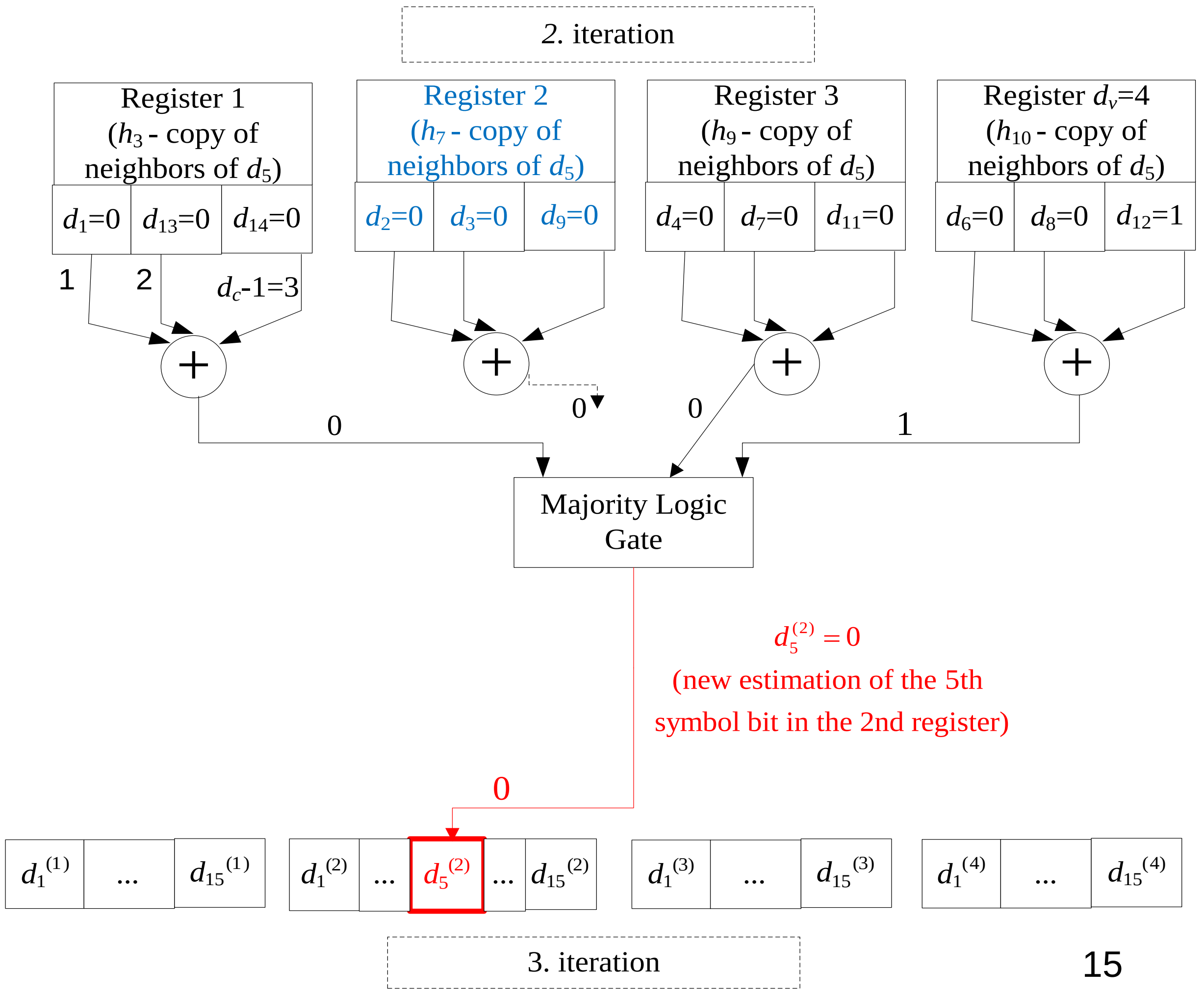
$$\sum_{e'\in E(12)\backslash\{d_{12}->h_{14}\}} m_2'(e') = m_2'(h_1->d_{12}) + m_2'(h_2->d_{12}) + m_2'(h_{10}->d_{12}) = 0 \;\Rightarrow\; m_2(d_{14}->h_{14}) = 0$$



d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} d_{11} d_{12} d_{13} d_{14} d_{15}
0
0
0
0
$d_v = 4$
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9 h_{10} h_{11} h_{12} h_{13} h_{14} h_{15}

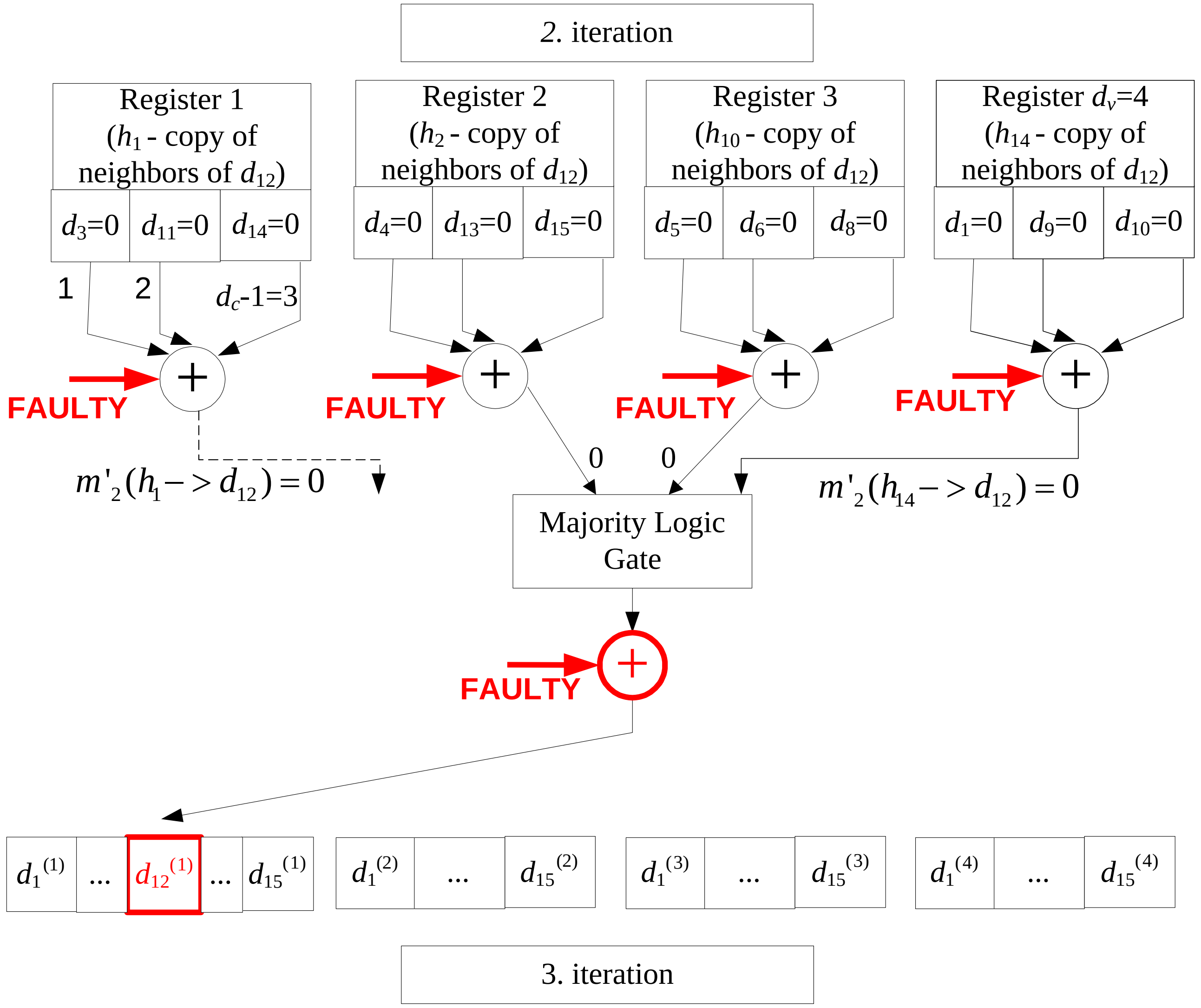
Taylor-Kuznetsov memory – updating bit 5 in the second register

$$
\mathbf{H} = \begin{array}{ccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}
$$



2. iteration
Register 1 (h_3 - copy of neighbors of d_5)
$d_1=0$
$d_{13}=0$
$d_{14}=0$
1
2
d_c-1=3
Register 2 (h_7 - copy of neighbors of d_5)
$d_2=0$
$d_3=0$
$d_9=0$
Register 3 (h_9 - copy of neighbors of d_5)
$d_4=0$
$d_7=0$
$d_{11}=0$
Register d_v=4 (h_{10} - copy of neighbors of d_5)
$d_6=0$
$d_8=0$
$d_{12}=1$
+
0
0
0
1
Majority Logic Gate
$d_5^{(2)}=0$
(new estimation of the 5th symbol bit in the 2nd register)
0
$d_1^{(1)}$ … $d_{15}^{(1)}$
$d_1^{(2)}$ … $d_5^{(2)}$ … $d_{15}^{(2)}$
$d_1^{(3)}$ … $d_{15}^{(3)}$
$d_1^{(4)}$ … $d_{15}^{(4)}$
3. iteration

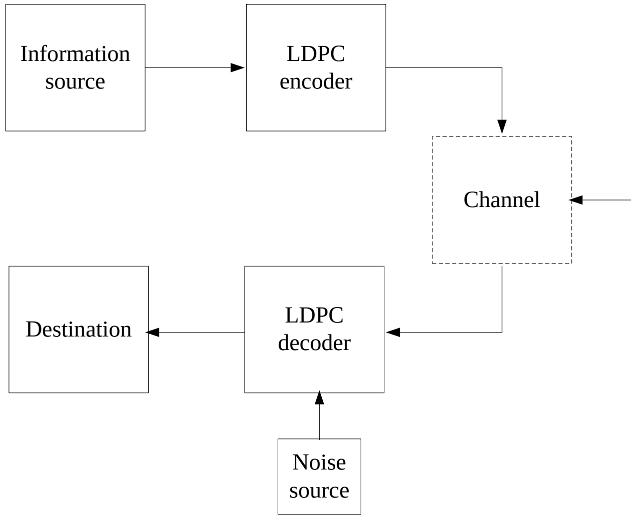
Taylor-Kuznetsov memory architecture with failures

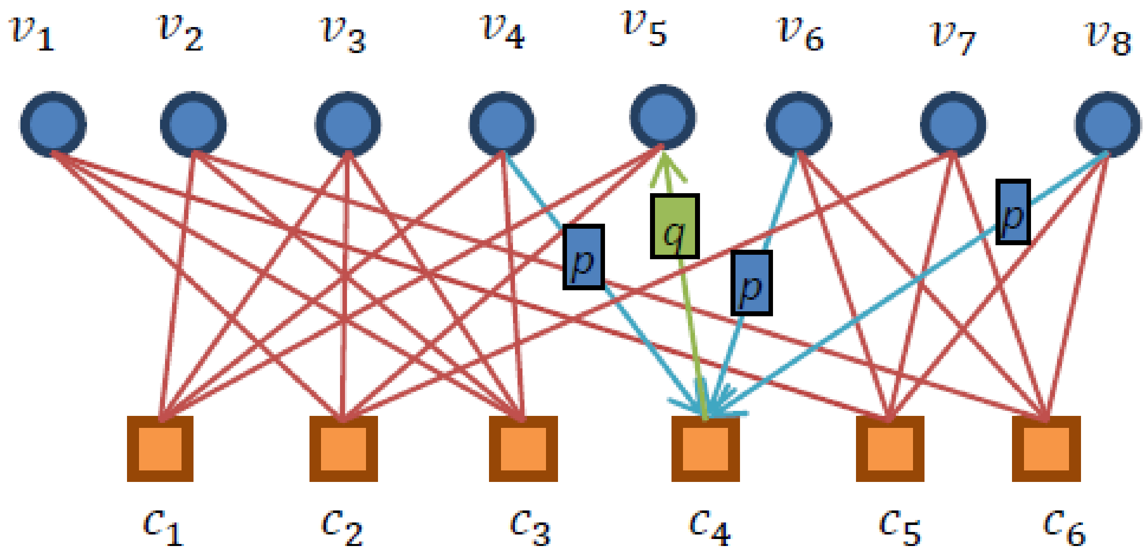


2. iteration
Register 1 (h_1 - copy of neighbors of d_{12})
d_3=0
d_{11}=0
d_{14}=0
1
2
d_c-1=3
Register 2 (h_2 - copy of neighbors of d_{12})
d_4=0
d_{13}=0
d_{15}=0
Register 3 (h_{10} - copy of neighbors of d_{12})
d_5=0
d_6=0
d_8=0
Register d_v=4 (h_{14} - copy of neighbors of d_{12})
d_1=0
d_9=0
d_{10}=0
FAULTY
FAULTY
FAULTY
FAULTY
$m'_2(h_1 -> d_{12}) = 0$
0
0
$m'_2(h_{14} -> d_{12}) = 0$
Majority Logic Gate
FAULTY
$d_1^{(1)}$ … $d_{12}^{(1)}$ … $d_{15}^{(1)}$
$d_1^{(2)}$ … $d_{15}^{(2)}$
$d_1^{(3)}$ … $d_{15}^{(3)}$
$d_1^{(4)}$ … $d_{15}^{(4)}$
3. iteration

Faulty Gallager-B

- Transient failures due to the noise or timing errors
- Decisions made in variable nodes and parity check nodes are unreliable (variable node is faulty with probability *p*, parity check node is faulty with probability *q*).



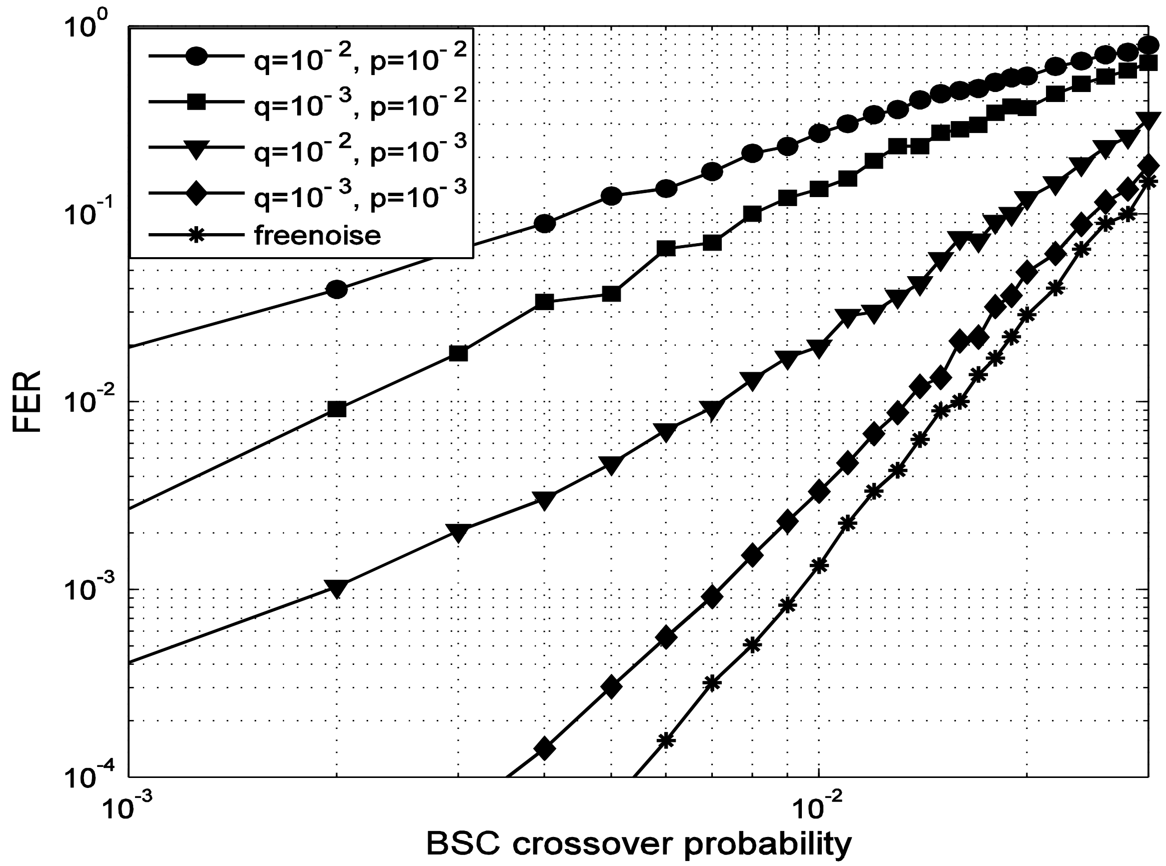


Faulty Gallager-B

- Let the failures are uncorrelated and data independent.
- Rate of number of incorrectly decoded codewords and number of transmitted codewords (frame error rate – FER) is determined for
  - variable crossover probability in BSC channel (or memory)
  - fixed probability of faulty in variable node, p
  - fixed probability of faulty in check node, q
  - fixed number of iterations during the decoding process
  - various QC codes are considered:
    - Tanner code (155,64) with n=155, ρ=5 and γ=3,
    - QC code with n=305, ρ =5 and γ =3,
    - QC code with n=155, ρ =5 and γ =4,

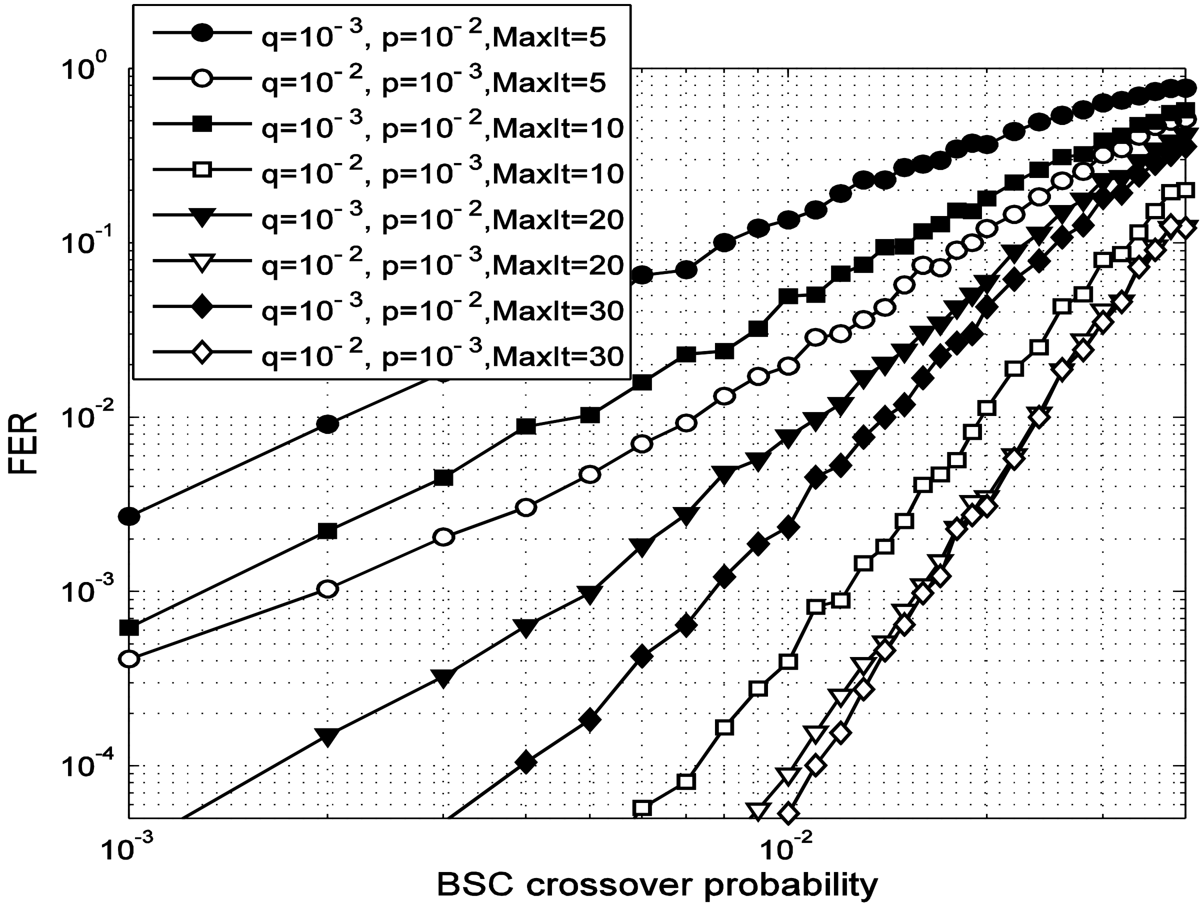
Faulty Gallager-B, Tanner (155,64), MaxIt=5

- Variable nodes are much more sensitive to processing noise and they should be realized by using more reliable hardware.



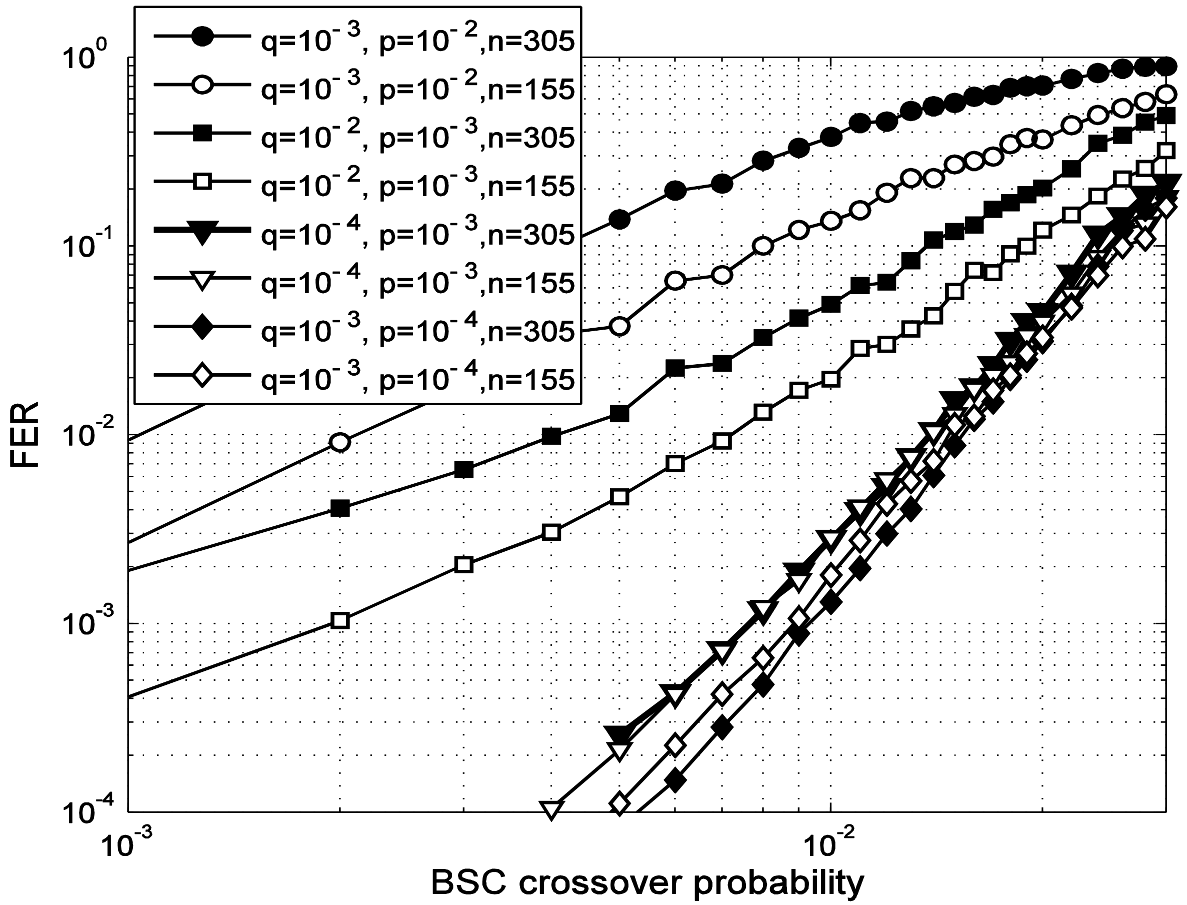
Faulty Gallager-B, iterations

- Performances can be improved by increasing the number of iterations, but variable nodes are still more critical.



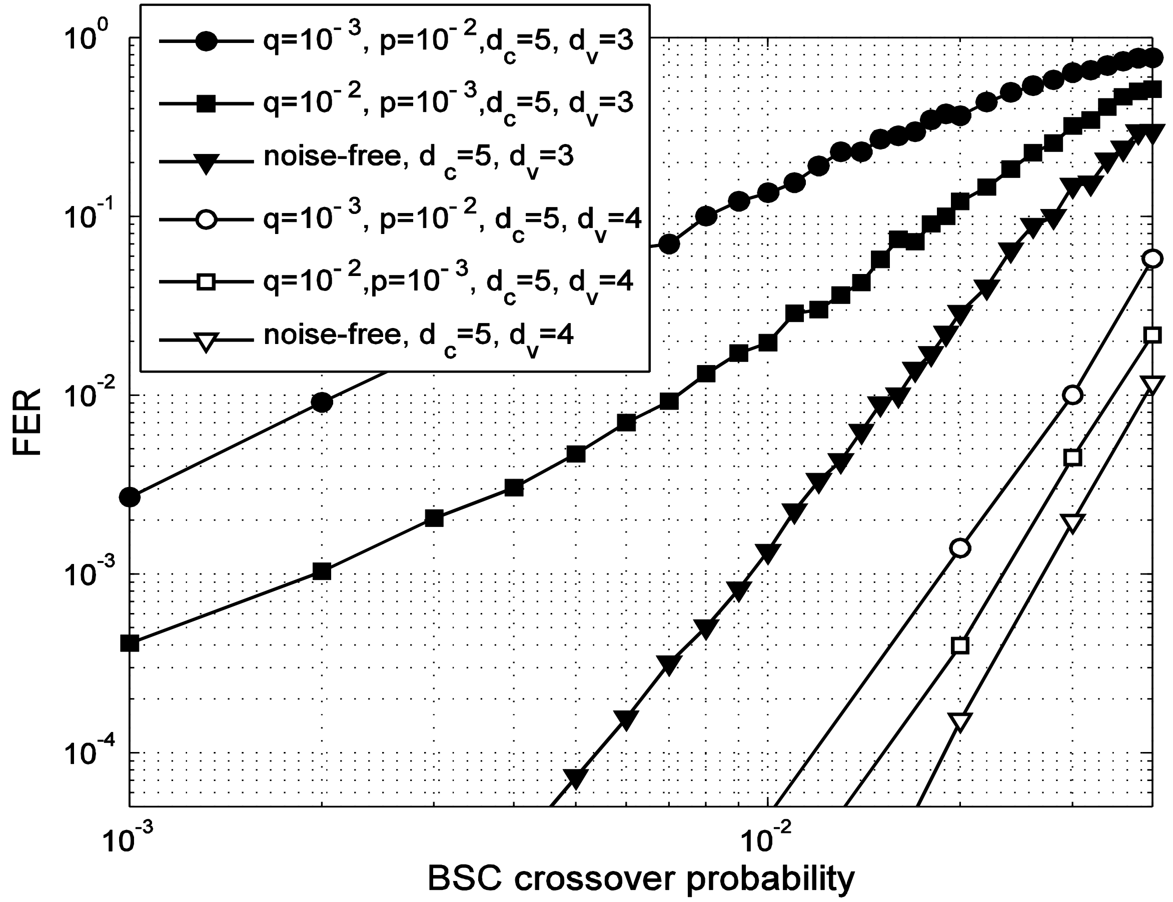
Faulty Gallager-B, codeword length

- Although the code with longer codewords has better correcting capabilities, it is also more prone to processing errors.



Faulty Gallager-B, impact of γ

- It is interesting to notice that performance of code with lower code rate are less degraded by decoder failures.



Parallel BF decoder

BSC

The channel output (as well as the input) has two levels.

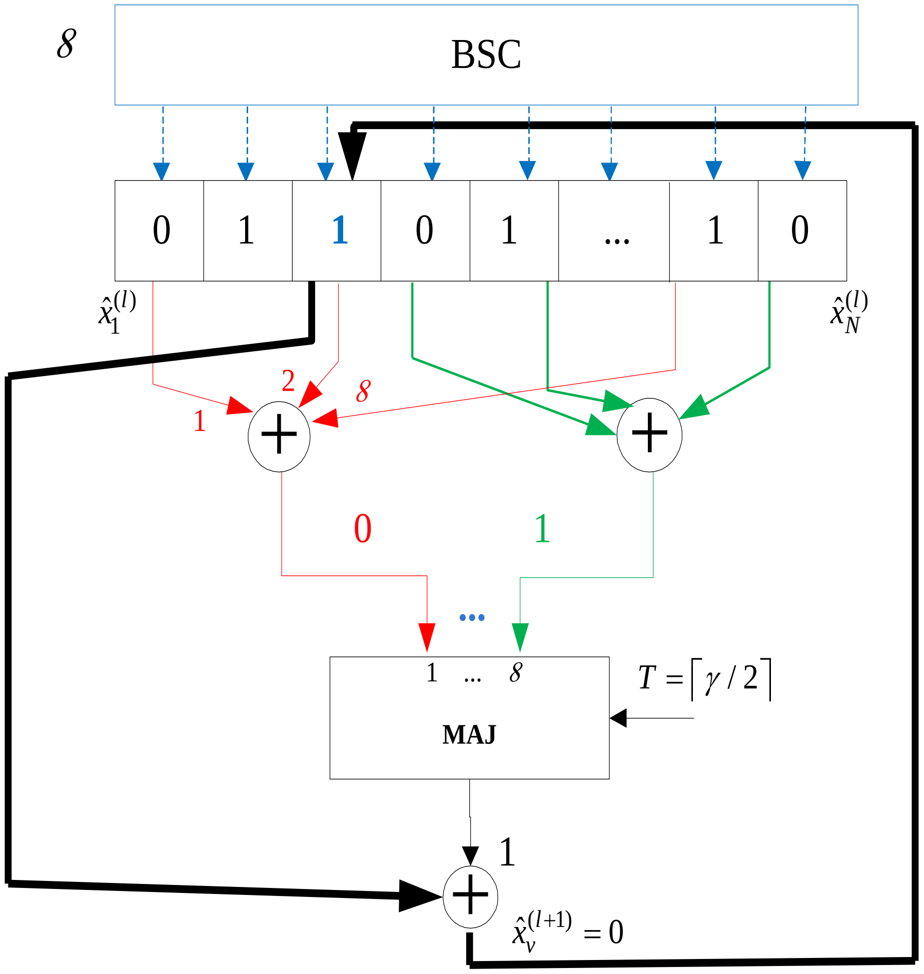
Crossover probability α.

For every variable node we check if it is satisfied:

$$\sum_{c \in N_v} \bigotimes_{u \in N_c} \hat{x}_v^{(l)} \leq \lceil \gamma / 2 \rceil$$

Valid codeword:

$$\sum_{c \in N_v} \bigotimes_{u \in N_c} \hat{x}_v^{(l)} = 0$$



GDBF for BSC

GBDF designed for AWGN

$$\square_v^{(l)} \square\ \chi_v^{(l)}\eta_v + \sum_{v\in N_v}\prod_{u\in N_c}\chi_u^{(l)}$$

Substitution:

$$\chi_v^{(l)} = 1-2\hat{x}_v^{(l)}, \quad \eta_v = 1-2y_v$$

$$\square_v^{(l)}(\mathbf{d}) \square\ 2-2(\hat{x}_v^{(l)}\oplus y_v)+\gamma_v-2\sum_{c\in N_v}\bigotimes_{u\in N_c}\hat{x}_u^{(l)} \rightarrow \min$$

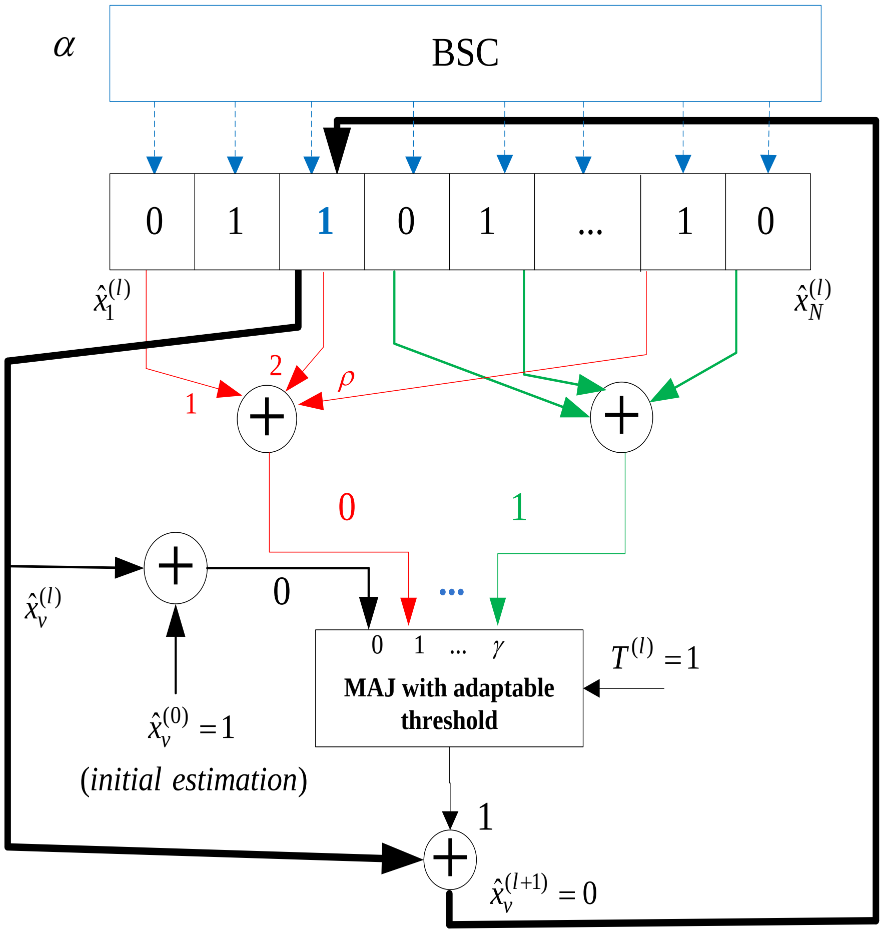
Modified inverse function

Irregular:

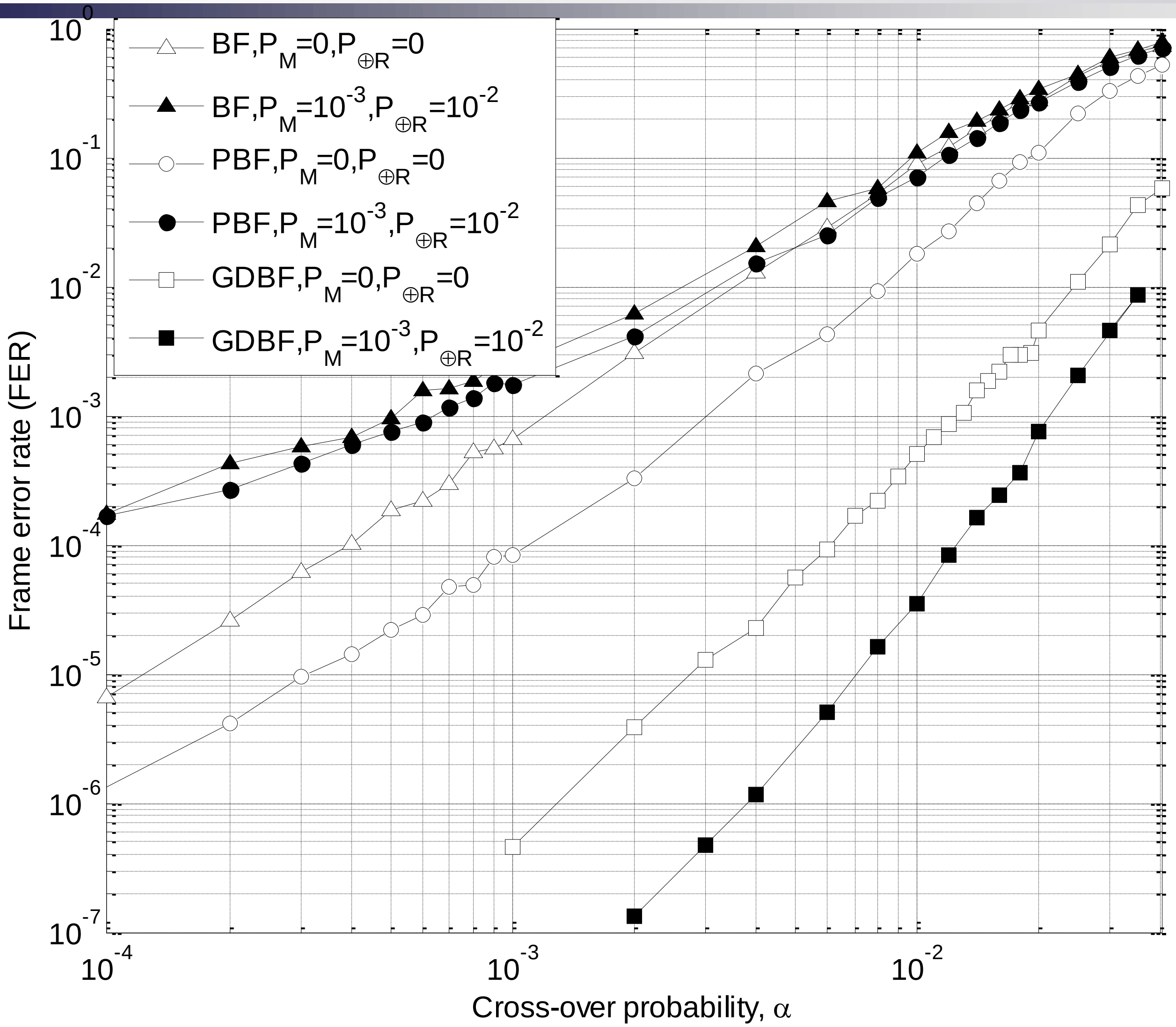
$$\Lambda_v^{(l)} \square\ 1-0.5\Delta_v^{(l)}$$
$$= \hat{x}_v^{(l)}\oplus y_v + \sum_{c\in N_v}\bigotimes_{u\in N_c}\hat{x}_u^{(l)} - 0.5\gamma_v$$

Regular:

$$\Lambda_v^{(l)} = \hat{x}_v^{(l)}\oplus y_v + \sum_{c\in N_v}\bigotimes_{u\in N_c}\hat{x}_v^{(l)}$$



Tanner (155,64), faulty BF/PBF/GDBF



PGDBF for BSC, faulty decoder (memory, XOR, ML)

Paralel bit-flipping:

$$\Lambda_v^{(l)}(\boldsymbol{d}) = \sum_{c \in N_v} \bigotimes_{u \in N_c} \hat{x}_v^{(l)}$$

Valid codeword:

$$\sum_{c \in N_v} \bigotimes_{u \in N_c} \hat{x}_v^{(l)} = 0$$

Probabibistic GDBF (PGDBF):

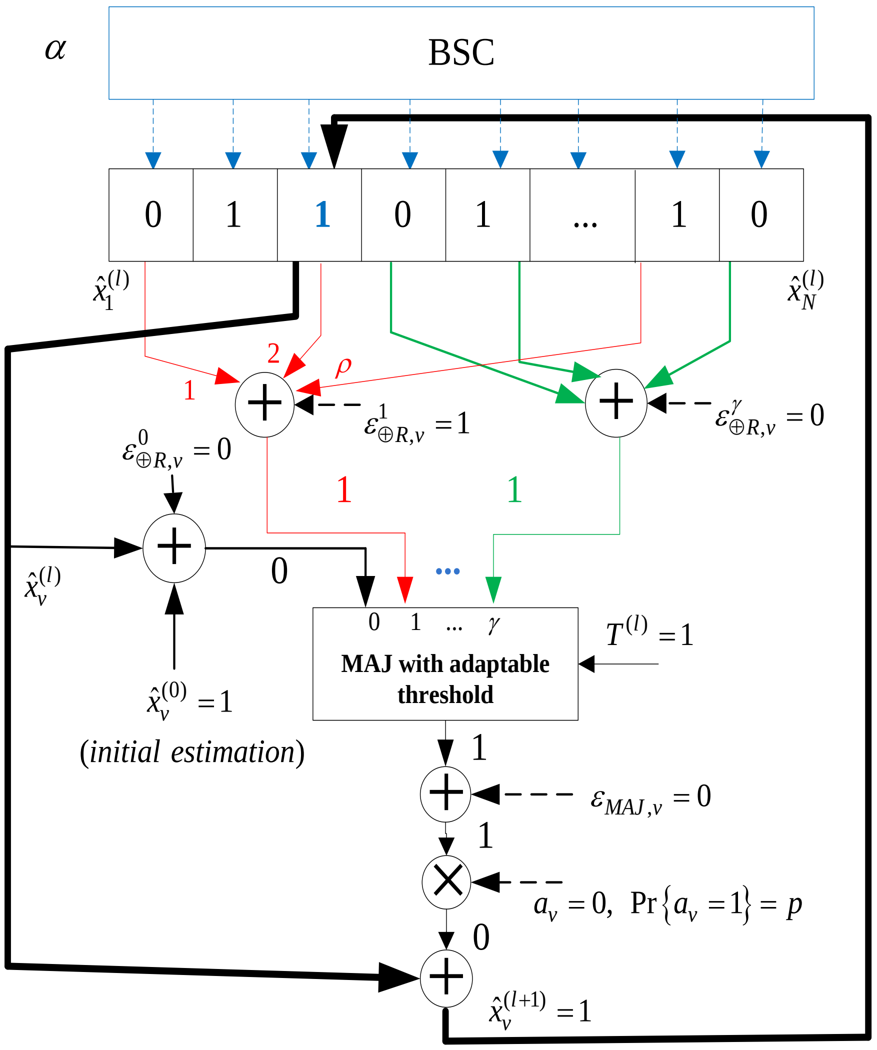
- Necessary condition for flipping

$$\Lambda_v^{(l)}(\boldsymbol{d}) \geq T^{(l)}$$

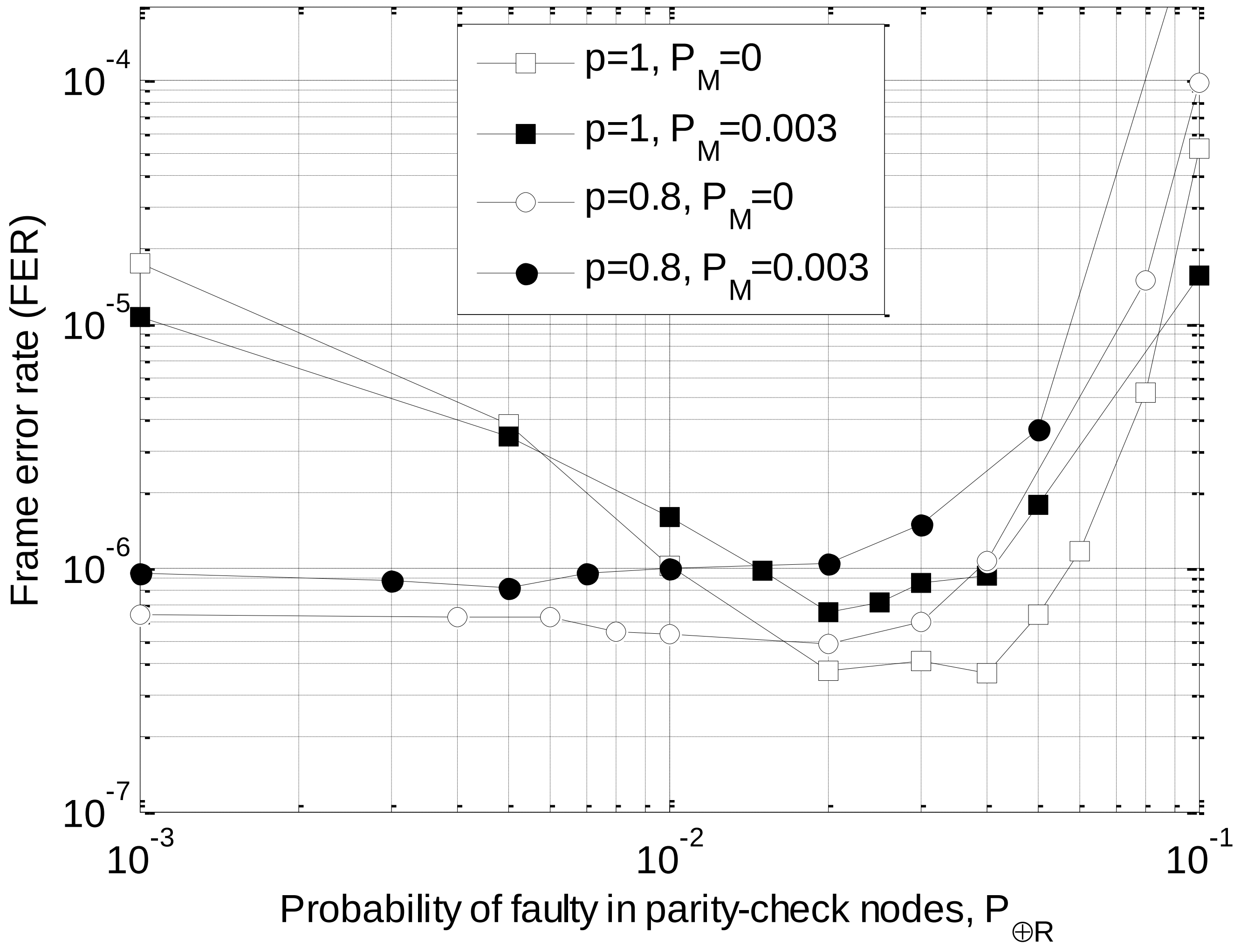
- One additional two-input AND gate
- Additional random number generator

Special cases:

- Parallel BF, probabilistic BF, GDBF for BSC (p=1)

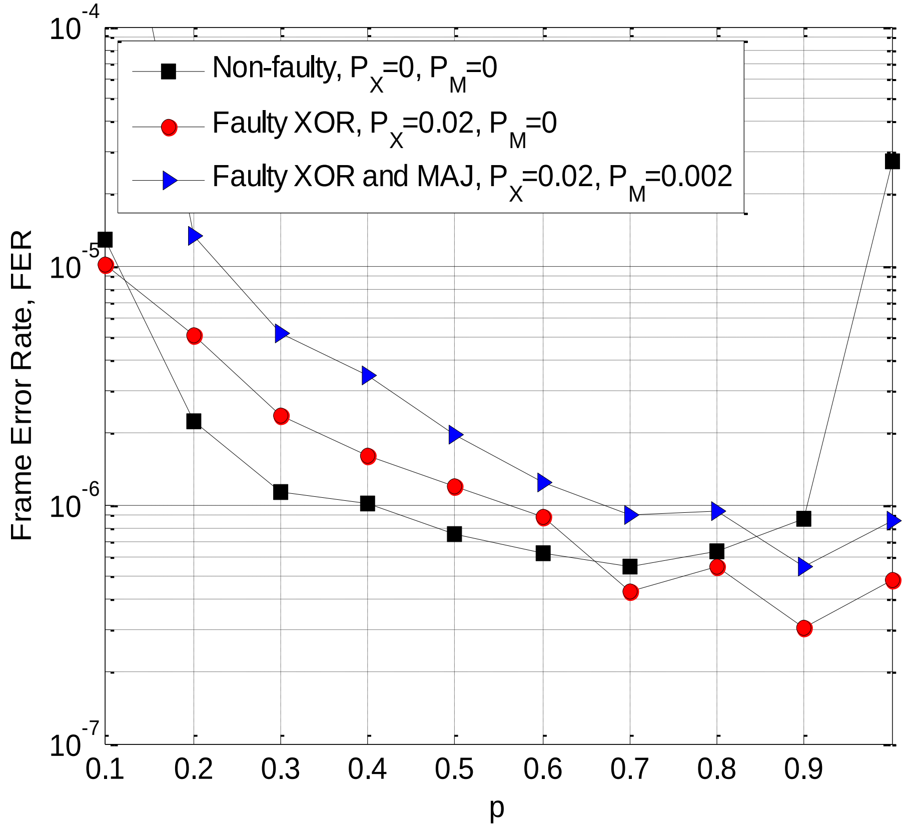

α
BSC
0 1 1 0 1 ... 1 0
$\hat{x}_1^{(l)}$
$\hat{x}_N^{(l)}$
1
2
ρ
$\varepsilon_{\oplus R,v}^{1} = 1$
$\varepsilon_{\oplus R,v}^{\gamma} = 0$
$\varepsilon_{\oplus R,v}^{0} = 0$
1
1
$\hat{x}_v^{(l)}$
0
0 1 ... γ
MAJ with adaptable threshold
$T^{(l)} = 1$
$\hat{x}_v^{(0)} = 1$
(initial estimation)
1
$\varepsilon_{MAJ,v} = 0$
1
$a_v = 0$, $\Pr\{a_v = 1\} = p$
0
$\hat{x}_v^{(l+1)} = 1$

Faulty XOR and memory, the impact of probabilism

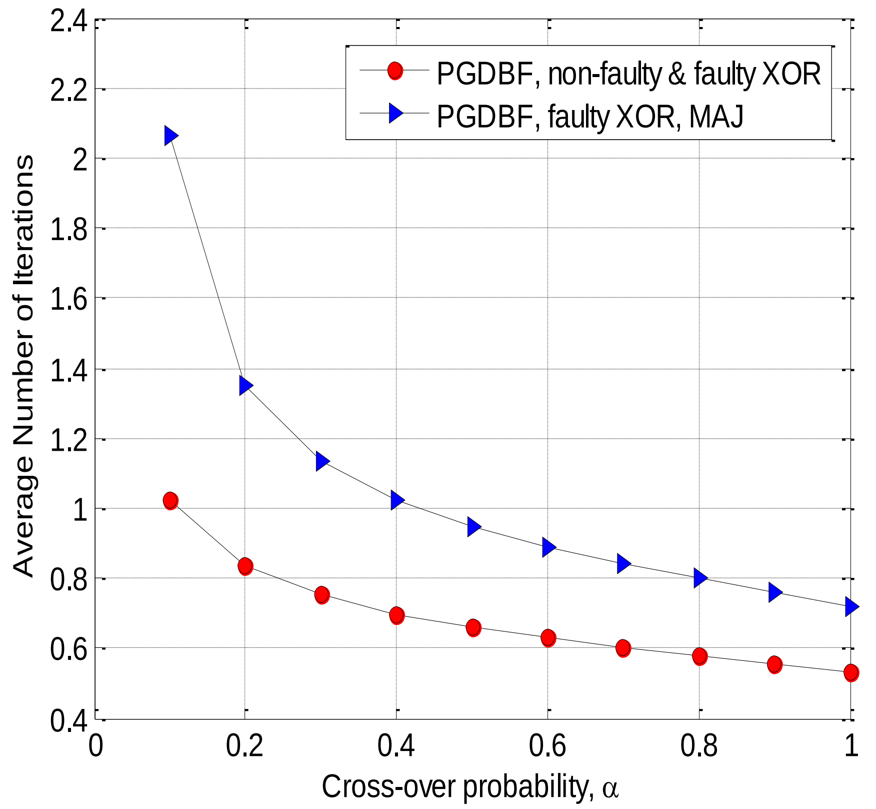


Tanner (155,64), faulty PGDBF, the impact of parameter *p* for $\alpha=4\times10^{-3}$

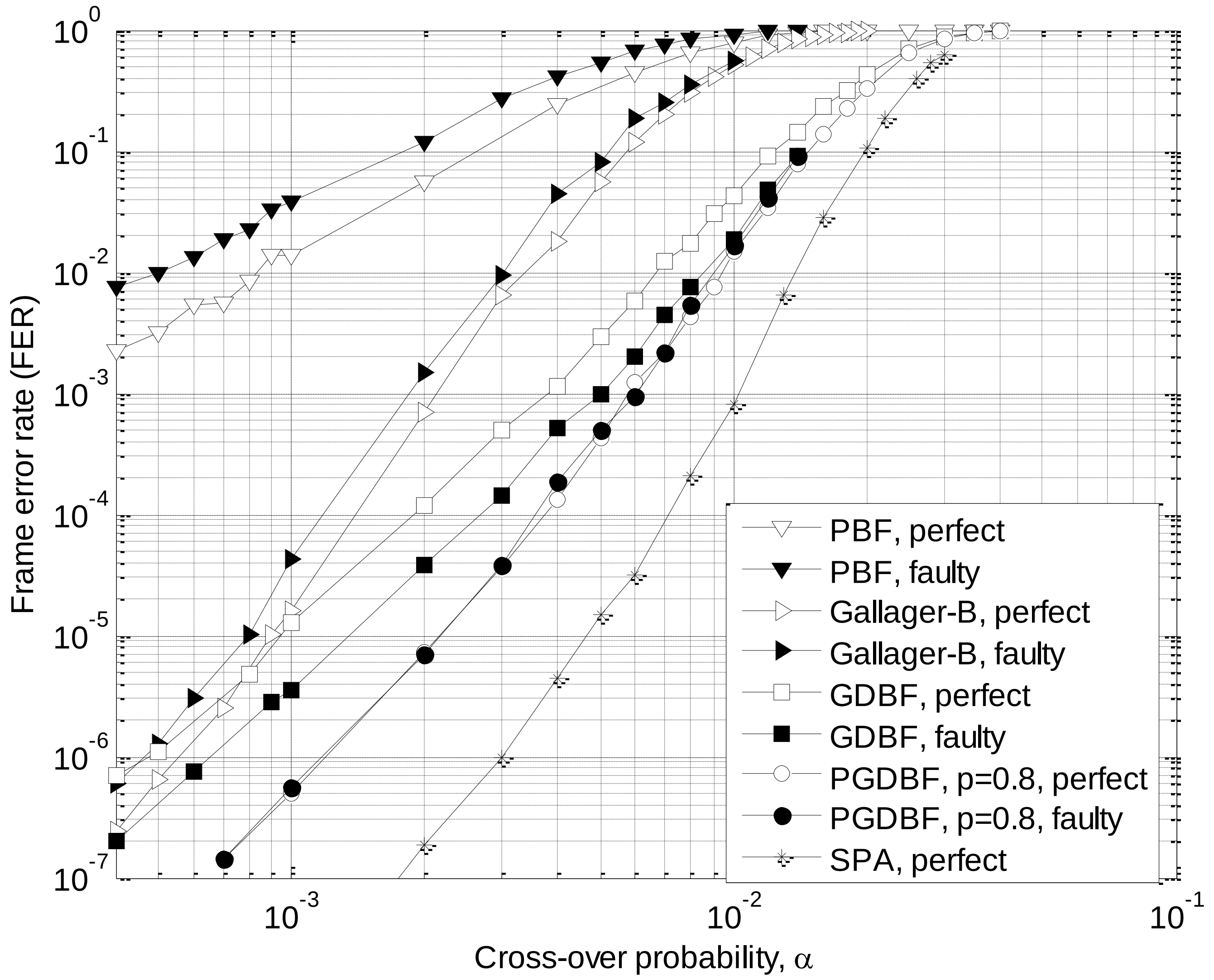
Frame Error Rate:



Average number of iterations:



QC-732, BF/PBF/GDBF/PGDBF/SPA



Conclusion

- **We considered:**

  - uncorrelated errors (von-Noyman type), different probability of injected failures in
    - check nodes (XOR gates), denoted by q
    - variable nodes (ML gates), denoted by p
  - data dependent failures – what if probability of injected failures depend of current (and previous) inputs of XOR gates and ML gates?

- **Hard decision decoders:**
  - Variable nodes are more critical than parity check nodes. Values of $q=1/n$, $p=q/10$ can be tolerated.
  - Gallager-B decoder outperforms BF algorithm and it is more immune to hardware failures.
  - In some cases GDBF work better in the presence of transient failures!
  - PGDBF has large immunity to hardware failures.

THANK YOU!