Åbo Akademi University

Digital Television – Lecture 5

Forward Error Correction (FEC)

Error Correction in Transmissions

- Need for error correction in transmissions
  - Loss of data during transmissions can make received data useless
- Different FEC methods in DVB:
  - Reed-Solomon codes (byte level, block code)
  - Convolutional codes (bit level, streaming code)
  - Low-Density Parity-Check codes (bit level, block code)
- FEC codes are combined with other methods to improve error correction performances on average
  - Interleaving (bit and byte levels)
  - Pseudo Random Bit Scrambling (bit level)

DVB Forward Error Correction

- Target: the transport stream should be Quasi Error Free (QEF) at the receiver
  - Bit Error Rate (BER) < 10^{-10}, i.e. 1 error on 10^{10} bits
- Compare with TCP:
  - Error free reception by resending packets
- Repetition code:
  - Transmit every bit e.g. 3 times. Receiver votes on the correct value
- Resending data not feasible solution in video streaming services and is in general very costly
- The repetition code is the worst possible error correction code existing

Basics for Error Correction

- Assume that the (source/information) bits at the transmitter can be grouped into words of k bits
- A FEC code C maps each possible sequence of the k bits onto unique *codewords* of length $n>k$ (2^k different codewords)
- In a *systematic* code, the source bits are contained sequentially within each codeword, either in the beginning or the end. This is not the case in a *non-systematic* code
- Example: k=2 and n=5

Systematic code

Source		Codewords
00		**00**000
01	⟶	**01**011
10		**10**101
11		**11**110

Non-systematic code

Source		Codewords
00		01000
01	⟶	00011
10		11101
11		10110

Basics for Error Correction

- For two bit vectors x and y of length n, the Hamming distance is defined as

$$d(x, y) := |\{i : x_i \neq y_i, 0 \leq i < n\}|$$

i.e., the number of bits that are different in x and y

x = 0111**0**0**0**1110
y = 0111**1**0**1**1110

$d(x,y) = 2$

- We try to find a code C that maximizes the distance between any two codewords in C, i.e., maximize the *minimum Hamming distance*

$$d_{min} = \min_{x,y \in C} \{d(x, y) | x \neq y\}$$

Basics for Error Correction

- The minimum Hamming distance allows us to determine the *minimum* error correction capability t of any block code

$$d_{min} \geq 2t + 1 \quad \longrightarrow \quad t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor$$

- A code C with codeword length n and information length k has a *code rate* defined as $r=k/n$
  - Fraction of the available bandwidth used for transmitting source data
  - Example: A code C with $r=1/2$ is used in a network with transmission rate 1 Mbps. Because of the code rate, useful information is received at a rate of 0.5 Mbps.
- We want to have a code C with as high code rate as possible and as large d_{min} as possible
  - The choice of code rate is a design parameter

Parity-Check Example

- Add a parity bit to the end of each sequence of *k* information bits

  011 **0**

  010 **1**

  101 **0**

  The value of the parity bit is chosen so that there is an even number of 1's in the bit sequence; corresponds to exclusive-OR of the bits
- The minimum Hamming distance between any two codewords in this type of a parity check code is 2
- In this systematic code, the codeword length is 4 bits of which 3 bits contain source information. A single bit error can now be *detected*, but not *corrected*
- The example code has code rate $r=3/4$, i.e., 75% of the available bandwidth is used for transmitting source data

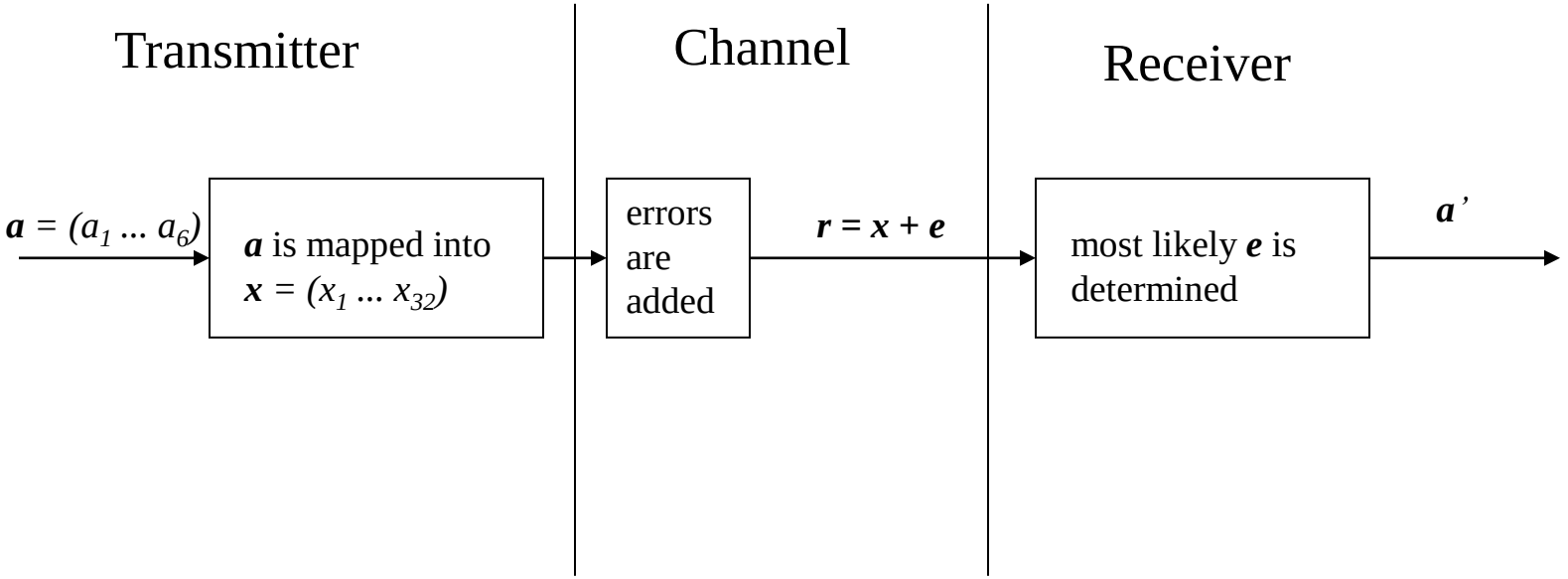
Error Correction

- Parity checking enables error detection. It is also possible to construct codes, that can perform error correction
- Idea: construct a code, where the Hamming distance between any two codewords is as large as possible.
- For any received word *r*, compare the received word with all codewords in the code, and select the codeword which has the smallest Hamming distance to *r*

Code		Distances	Decoded
01001	Received	2	
01110		1	01110
10011	01111	3	
11000		4	

Forward Error Correction

- The term *Forward* in Forward Error Correction comes from the fact that the error correction is in a sense done in advance, i.e., parity data is added prior to transmission
- FEC coding only makes sense if the bit error rate while using coding is less than without coding → *coding gain*



Parity-Check Codes

- We have seen that codes with a single parity bit at the end of each codeword has minimum Hamming distance 2.
  - Such a code can only detect errors
- Suppose we have 9 source bits $\boldsymbol{a}=(a_1,\ldots, a_9)$, and arrange them in a square. Add a parity bit to each column and each row

$$\begin{array}{ccc|c} a_1 & a_2 & a_3 & p_1 \\ a_4 & a_5 & a_6 & p_2 \\ a_7 & a_8 & a_9 & p_3 \\ \hline p_4 & p_5 & p_6 & \end{array} \longrightarrow \begin{array}{ll} a_1 \oplus a_2 \oplus a_3 = p_1 & a_1 \oplus a_4 \oplus a_7 = p_4 \\ a_4 \oplus a_5 \oplus a_6 = p_2 & a_2 \oplus a_5 \oplus a_8 = p_5 \\ a_7 \oplus a_8 \oplus a_9 = p_3 & a_3 \oplus a_6 \oplus a_9 = p_6 \end{array}$$

$\oplus$ denotes exclusive-OR

- This gives us the possibility to detect and correct errors

Parity-Check Codes

- The set of equations for the example code can be rewritten as

$$a_1 \oplus a_2 \oplus a_3 \oplus p_1 = 0 \qquad a_1 \oplus a_4 \oplus a_7 \oplus p_4 = 0$$
$$a_4 \oplus a_5 \oplus a_6 \oplus p_2 = 0 \qquad a_2 \oplus a_5 \oplus a_8 \oplus p_5 = 0$$
$$a_7 \oplus a_8 \oplus a_9 \oplus p_3 = 0 \qquad a_3 \oplus a_6 \oplus a_9 \oplus p_6 = 0$$

- Define the codeword vector as $\boldsymbol{x}=(a_1,\ldots, a_9, p_1,\ldots, p_6)$, and express the system of equations with the *parity-check* matrix $\boldsymbol{H}$.
- A *valid codeword* satisfies $\boldsymbol{H}\boldsymbol{x}^T=\boldsymbol{0}$
  - Summations are done modulo 2; corresponds to exclusive-OR

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Parity-Check Codes

- In the *parity-check* matrix ***H***, each row corresponds to a parity-check constraint and each column corresponds to the parity-check constraints that a given bit participates in
- The minimum Hamming distance can be determined from the parity-check matrix:

  A parity-check code C has minimum Hamming distance d if and only if any d-1 columns of its parity-check matrix are linearly independent but some d columns are linearly dependent

- The example code on the previous slide has d_{min}=3, i.e., it is guaranteed to correct 1 bit error.
  - If more errors than 1 occur, we cannot predict if the code can correct them or not

Towards Good Codes

- In order to obtain good parity-check codes, we need to increase the minimum Hamming distance
- To do this, we need to add more 1's to the parity-check matrix ***H***, in order to decrease the linear dependencies between the columns
- In general, each column should have a weight (number of non-zero entries) of at least 2
  - The identity sub matrix at the right side of the parity-check matrix in the example code is one of the major reasons to its low minimum Hamming distance

Low-Density Parity-Check Codes

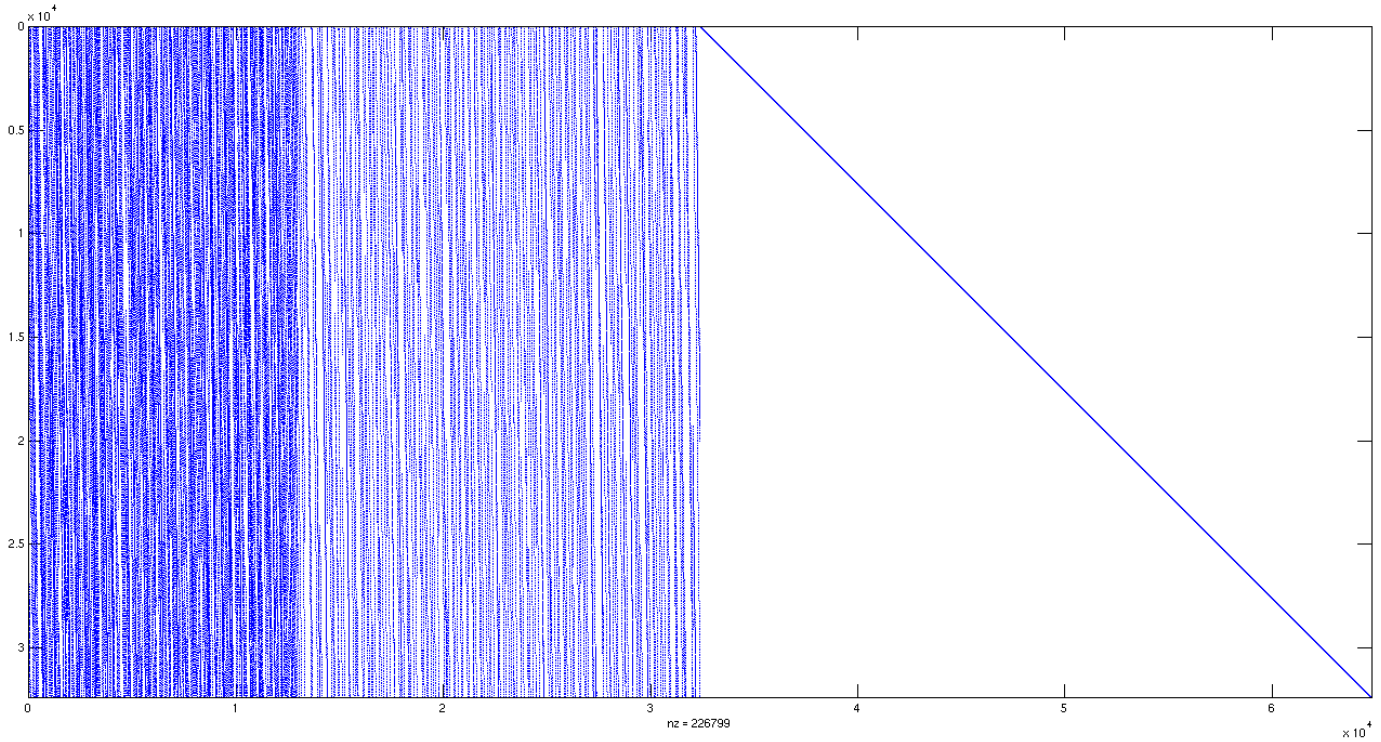
- LDPC codes are one of the best existing FEC codes
  - DVB-T2, DVB-S2
- The code length n is typically very large
  - In DVB-T2 and DVB-S2: n=64 800 or n = 16200 bits
- LDPC codes are *sparse*, i.e., the fraction of non-zero entries in $\boldsymbol{H}$ goes towards zero when $n \rightarrow \infty$
- The parity-check matrices are often generated at random, according to constraints on the distributions of the column and row weights
- Example: in an LDPC code, 1/3 of the columns should have weight 3, 1/3 of the columns should have weight 2, and 1/3 of the columns should have weight 1, while each row should have weight 4

DVB-T2 LDPC Parity-Check Matrix

Systematic code with $n = 64\,800$, $k = 32400 \rightarrow r = 1/2$

n columns

n-k rows



Encoding

- A code *C* maps the source bits to unique codewords
- When the information length *k* is large, it is impossible to maintain a list of all codewords of the code
  - A code with k=5000 has $2^k \approx 1.4*10^{1505}$ codewords, where each codeword is *n* bits
- Codewords are computed based on the source bits at the encoder
- In some cases, the codewords can be computed directly using the parity-check matrix
  - Systematic codes, where the sub matrix for the parity bits is lower or upper triangular
- In general, however, a *generator matrix* $\boldsymbol{G}$, needs to be determined for encoding purposes
  - Instead of storing all possible codewords in memory, we only need to store the generator and parity-check matrices in memory

Encoding

- If we can (through simple row and column operations) bring $\boldsymbol{H}$ into the form $\boldsymbol{H}$=[$\boldsymbol{P}$ | $\boldsymbol{I}$], where $\boldsymbol{P}$ is a random sub matrix, and $\boldsymbol{I}$ is the identity matrix corresponding to the parity columns, then the generator matrix $\boldsymbol{G}$ is obtained as

$$\boldsymbol{G}=[\ \boldsymbol{I}\ |\ \boldsymbol{P}^T\ ]$$

- From the source bits $\boldsymbol{a}=(a_1,\ldots, a_k)$, the codeword $\boldsymbol{x}$ is obtained as $\boldsymbol{x}=\boldsymbol{aG}$
- Example: k=2 and n=5

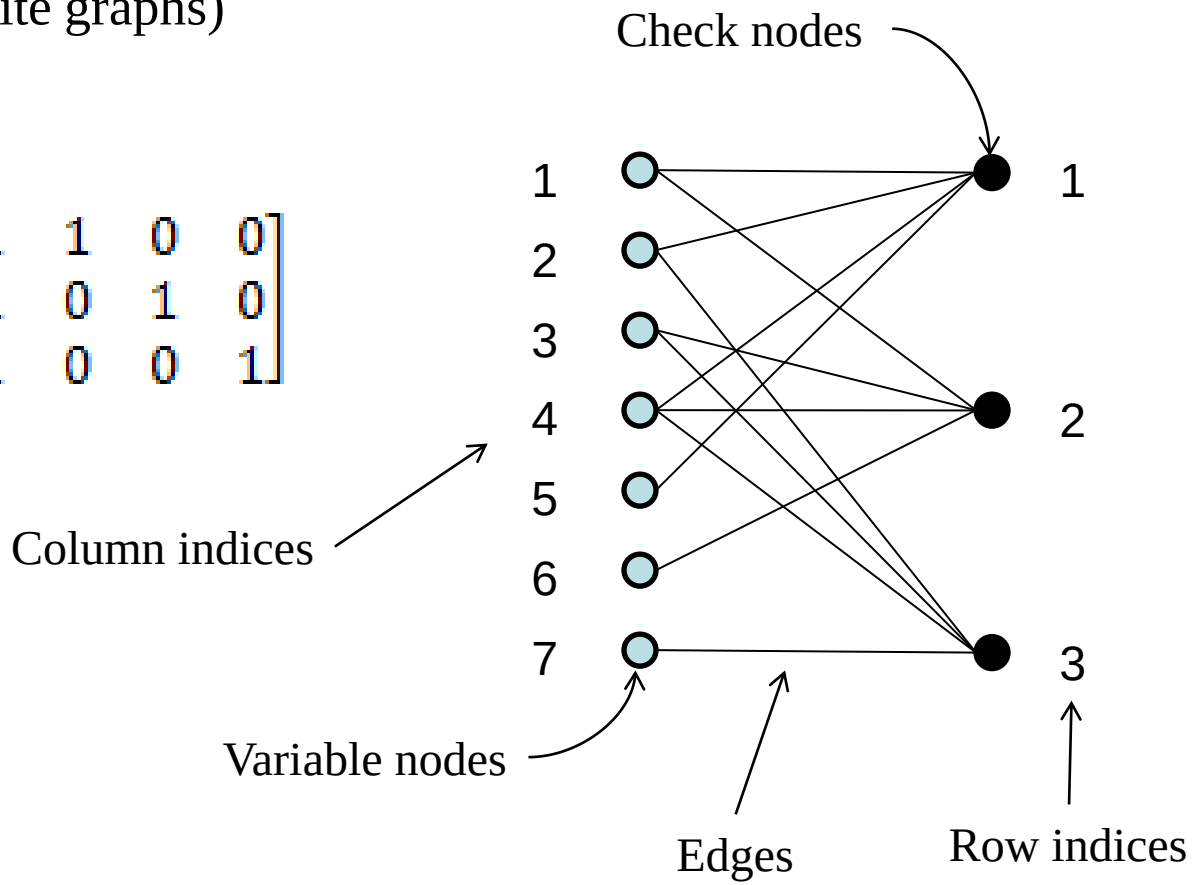
$$H = \left[\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}\right] \longrightarrow G = \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

$\boldsymbol{a}$=(0 1) $\rightarrow$ $\boldsymbol{x}$ = $\boldsymbol{aG}$ = [0 1 1 1 0] and $\boldsymbol{Hx}^T=\boldsymbol{0}$

Tanner Graph Representation

- The parity-check matrix can be visualized using Tanner graphs (bipartite graphs)

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$



Decoding of LDPC Codes

- Decoding is an iterative message passing process
- Initially, only *beliefs* of the bit values are available at the decoder
- The beliefs propagate through the graph → *Belief-Propagation* (BP) decoder
- One decoding iterations is as follows
  - Each variable node sends its belief of the bit value over all its edges to the corresponding check nodes
  - The check nodes compute the most likely bit values for all their neighbours (variable nodes they are connected to) and send these beliefs back
  - The variable nodes update their beliefs on the bit values using the incoming messages
- The decoder continues to iterate until a valid codeword has been found, or a maximum number of iterations has been performed

Decoding of LDPC Codes

- In DVB-T2 and DVB-S2, the LDPC codes are used so early in the communication chain that only likelihoods about the bit values are available to the decoder

- Each bit x_i is assigned a *log-likelihood ratio* (LLR) prior to decoding, defined as

$$LLR(x_i|y_i) = \log\left(\frac{\mathrm{P}(x_i = 0|y_i)}{\mathrm{P}(x_i = 1|y_i)}\right)$$

  where y_i is received information for bit i
- The magnitude of the LLR represents the probability of the bit having a certain value
- The sign of the LLR tells us if the probability of the bit value leans towards 0 or 1

Decoding Example

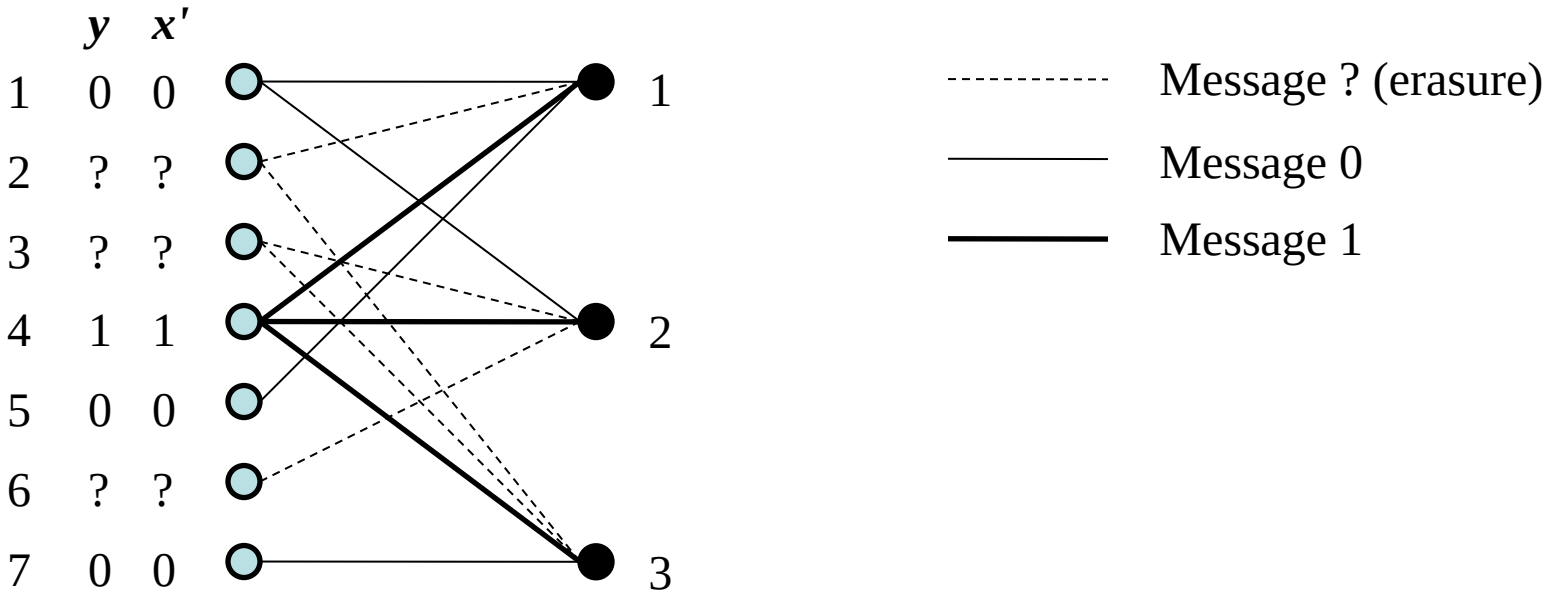
- Assume that bits are either received correctly or known to be erroneous (erasures)
  - This channel model is known as the Binary Erasure Channel (BEC)

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

- The codeword $\boldsymbol{x}$ = (0 1 0 1 0 1 0) is transmitted
- The received word is $\boldsymbol{y}$ = (0 ? ? 1 0 ? 0), where ? denotes an erasure (known error)
  - Greatly simplified example w.r.t. decoding in DVB. Here, messages passed will have values 0, 1, or ?, but in practice each message passed is an LLR value, i.e., floating point value.
  - The main decoding procedure is still the same

Decoding Example

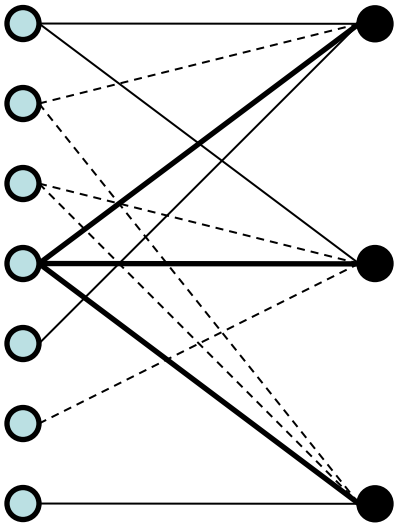
- Step 1: Variable to check
  - The variable nodes send their beliefs on the bit values to the check nodes, i.e., the beliefs are known to be correct or erasures because of the BEC
  - ***y*** is the received codeword and ***x'*** is the estimated codeword



Decoding Example

- Step 2: Check to Variable
  - The check nodes process the incoming messages, such that the constraints should be fulfilled
  - $\boldsymbol{Hx}^T$ should be 0, i.e. the sum modulo 2 of all bit values participating in all parity-check constraints (checks nodes) should be 0

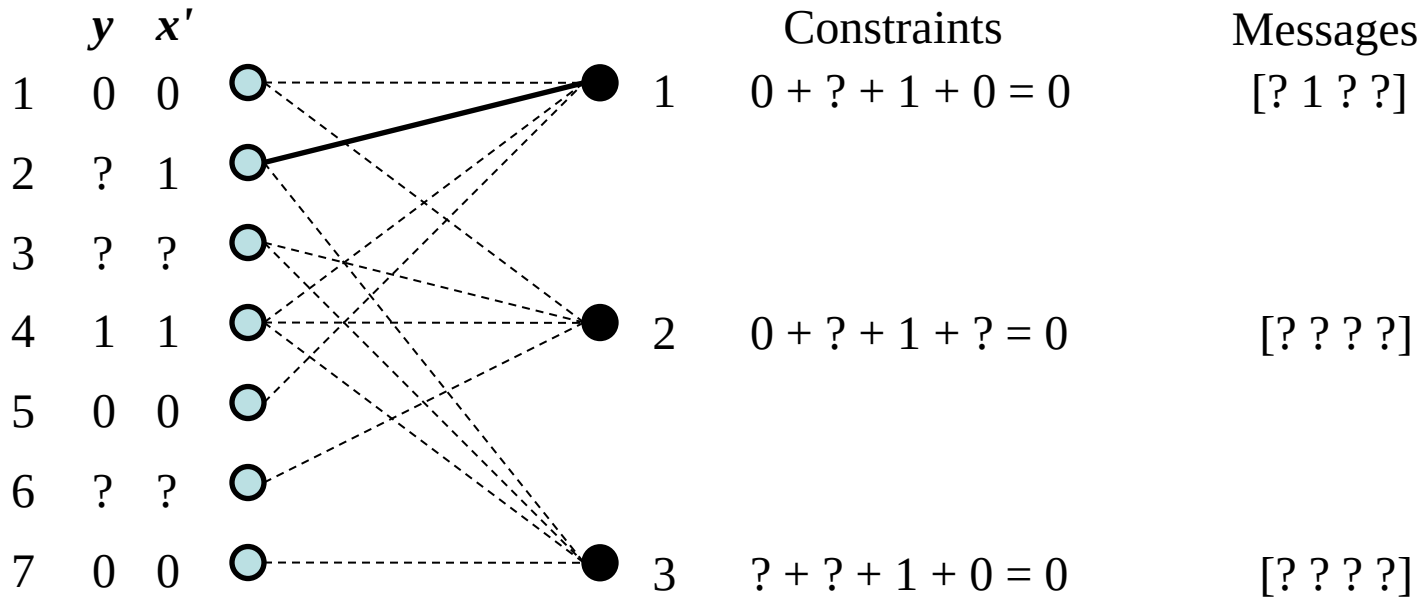
	y	x'
1	0	0
2	?	?
3	?	?
4	1	1
5	0	0
6	?	?
7	0	0



Constraints	
1	0 + ? + 1 + 0 = 0
2	0 + ? + 1 + ? = 0
3	? + ? + 1 + 0 = 0

Decoding Example

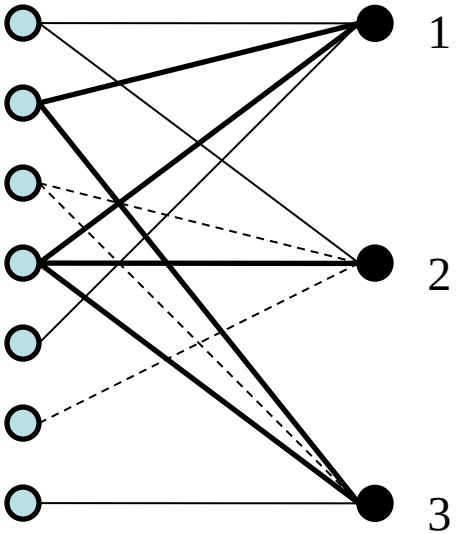
- Step 2: Check to Variable (continued)
  - Each check node sends a message back to each of its variable nodes
  - A message from a check node is equal to the sum modulo 2 of all incoming message, excluding the message that came from the variable node to where this message is being sent



Decoding Example

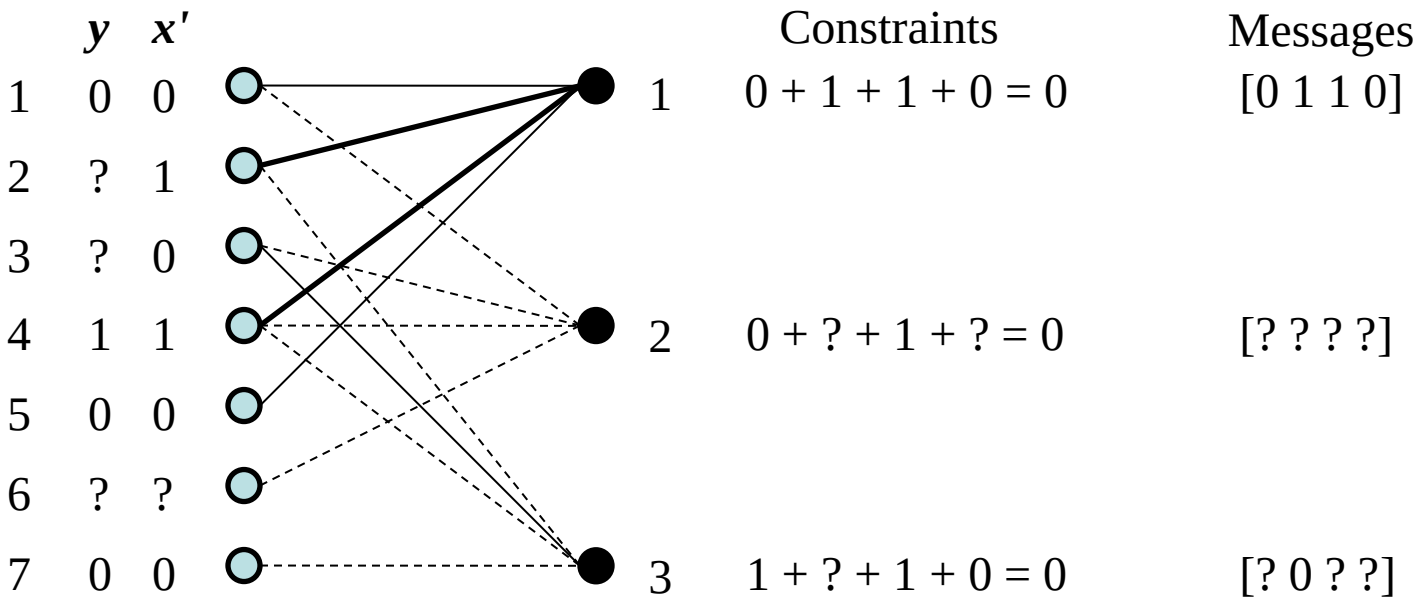
- Step 1: Variable to check
  - The variable nodes sends their *updated* beliefs on their bit values to the check nodes
  - At a variable node, the outgoing message is an erasure if *all* incoming messages are erasures *and* the received messages is an erasure

	y	x'
1	0	0
2	?	1
3	?	?
4	1	1
5	0	0
6	?	?
7	0	0



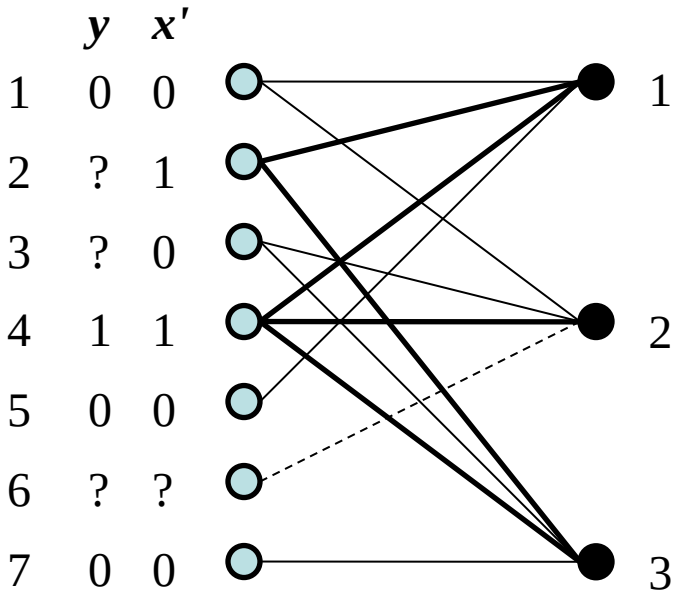
Decoding Example

- Step 2: Check to Variable
  - Each check node sends back their beliefs on the symbol values to each of its variable nodes, which satisfy their constraints



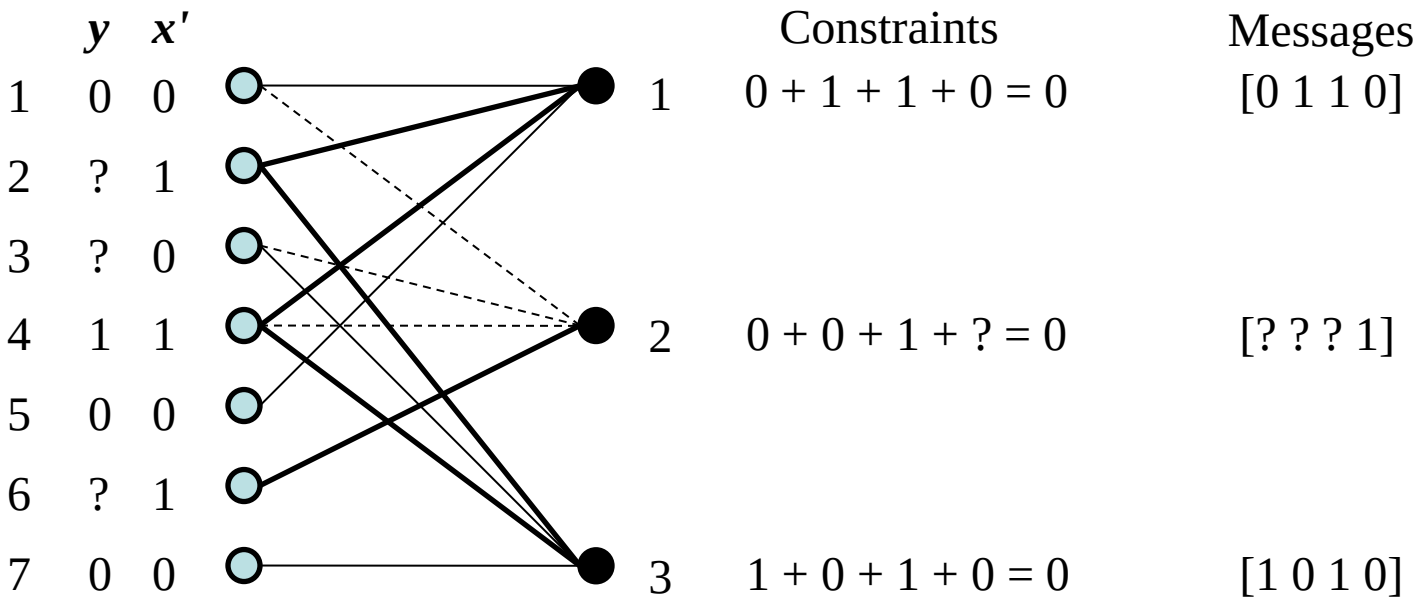
Decoding Example

- Step 1: Variable to check
  - The variable nodes sends their *updated* beliefs on their bit values to the check nodes



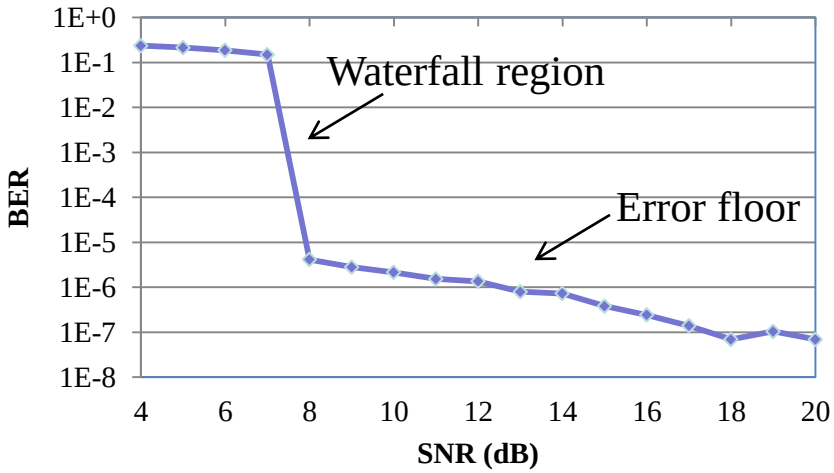
Decoding Example

- Step 2: Check to Variable
  - Each check node sends back their beliefs on the bit values to each of its variable nodes, which satisfy their constraints
- Decoding completed, i.e. original codeword recovered



LDPC Code Performance

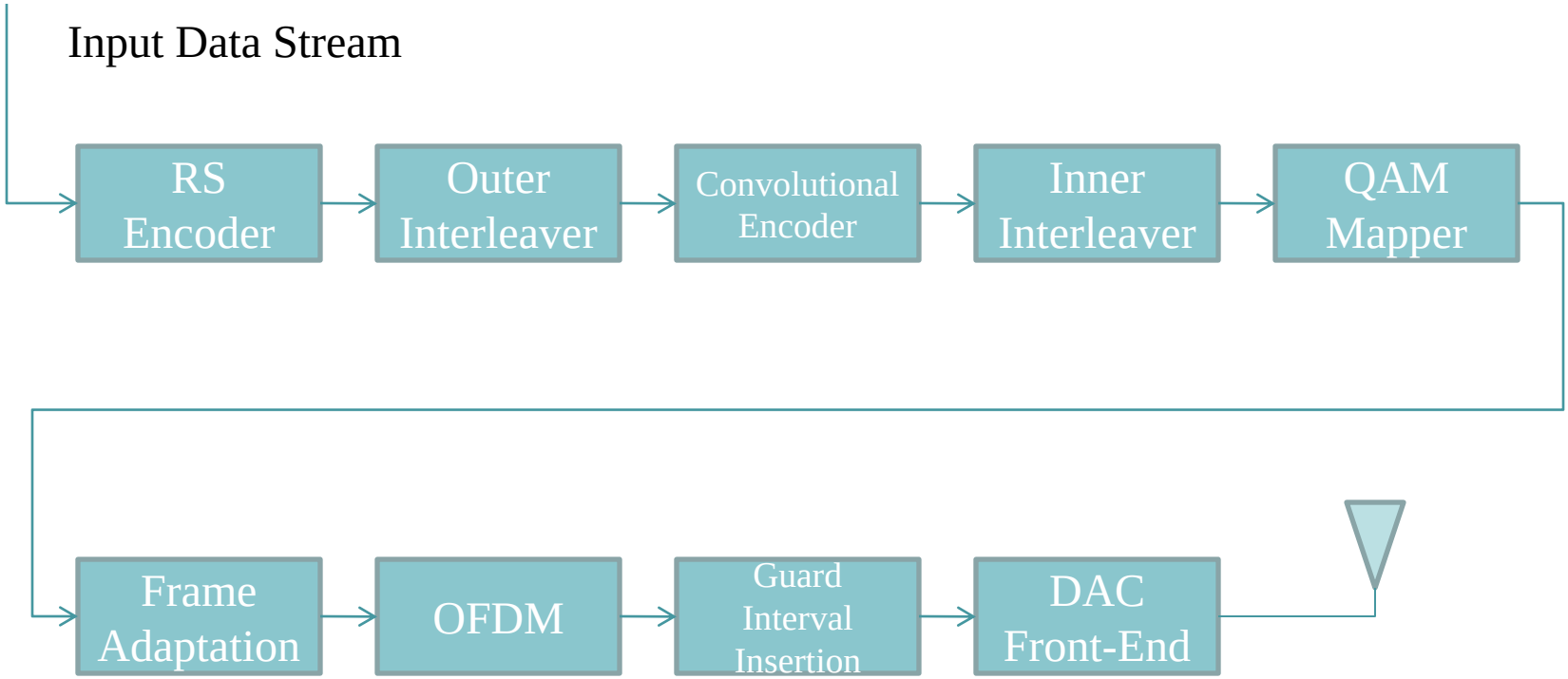
- Although the minimum Hamming distance is commonly used to measure code performance, in the case of LDPC codes, this measure is not adequate.
  - LDPC codes typically have far better error correcting performance than what can be derived from the minimum Hamming distance
- Well designed LDPC codes have performances that are very close to what is theoretically possible to achieve in communications (Shannon limit)



Coding in DVB

- LDPC codes produce small error floors
  - Occasional bit errors when one would expect zero BERs
- In second generation DVB standards, BCH (Bose-Chaudhuri-Hocquenghem) codes are used to correct any bit errors that may be present after LDPC decoding
  - The BCH codes have an error correction capability of roughly 12 bits in the BCH codewords of lengths 30 000 – 50 000 bits
- In first generation DVB standards, the coding scenario is different
  - Instead of LDPC codes, convolutional codes (less powerful) are used
  - Concatenated with Reed-Solomon codes, where each RS code protects a TS packet (RS codes operate on *bytes* instead of bits)
    - $n = 204$, $k = 188$, error correction capability 8 *bytes*
- The second generation DVB coding scheme is significantly better than the one in the first generation DVB standards

DVB-T Transmitter Block Diagram



DVB-T2 Transmitter Block Diagram