University of Stavanger

Data Link Layer

Coding and Error Control

Gianfranco Nencioni

Some slides are based on early ones prepared by Prof. Erdal Cayirci

University of Stavanger

Outline

- Error Detection
- Automatic Repeat Request (ARQ)
- Error Correction
  - Hamming Codes
  - Cyclic Codes
  - Low-Density Parity Codes
  - Polar Codes
  - Convolutional Codes
  - Turbo Codes
- Hybrid ARQ

Is error control needed?

- **Frame**: one of more contiguous sequences of bits

- P_b: Prob. of a single-bit error (a.k.a. bit error rate - BER)
- P_1: Prob. of no bit errors in the frame
- P_2: Prob. of one or more errors in the frame

F: number of bits per frame

$$P_1 = (1 - P_b)^F$$

$$P_2 = (1 - P_1)$$

Example: $P_b=10^{-5}$ $F=2046$ $P_2=0.02$

Need of error control!

Error Control

- Forward Error Correction (FEC)
  - Error Correction Codes
- Backward Error Correction (BEC)
  - Error Detection
  - Acknowledgments and Retransmissions
    - a.k.a. Automatic Repeat Request (ARQ)
    - *Note*: Data Link Layer and Transport Layer
- Reliable Delivery
  - **Error Correction**
  - Sequence Control
  - Flow Control

University of Stavanger

Error Detection

Parity Check

- Data block of k bits
- Error detection code of $n - k$ bits, where $(n - k) < k$
  - Check bits
- Frame of n bits

- In the parity check, $n - k = 1$, which is called **parity bit**

  Example (character transmission): The value of the parity bit is selected so that the character has even number of 1s (even parity) or an odd number of 1s (odd parity)
- **Undetected error**: if two (or any even number) of bits are inverted

Cyclic Redundancy Check (CRC)

University of Stavanger

- Check bits are called **Frame Check Sequence (FCS)**
- The resulting frame is exactly divisible by some predetermined number
- Procedure can be presented in three ways:
  - Modulo 2 Arithmetic
  - Polynomials
  - Digital Logic

Cyclic Redundancy Check (CRC)

University of Stavanger

Data block: ***D*** = 1101011011 (10 bits)

Pattern: ***P*** = 10011 (5 bits)

FCS: ***R*** = *to be calculated* (4 bits)

XOR

IN	OUT
0 0	0
0 1	1
1 0	1
1 1	0

```
11010110110000
10011
 10011
 10011
  10110
  10011
   10100
   10011
    1110
```

Remainder

Transmitted Frame:
1101011011 1110

University of Stavanger

Automatic Repeat Request (ARQ)

University of Stavanger

Automatic Repeat Request (ARQ) Process

- Used in both Data Link Layer and Transport Layer
- Based on an error detection code (e.g. CRC)
- Flow Control and Error Control

Flow Control: assuring that a transmitting entity does not overwhelm a receiving entity with data

Error Control: detecting and correcting errors that occur in the transmission of data

Automatic Repeat Request (ARQ) Process

Block of data = Protocol Data Unit (PDU)

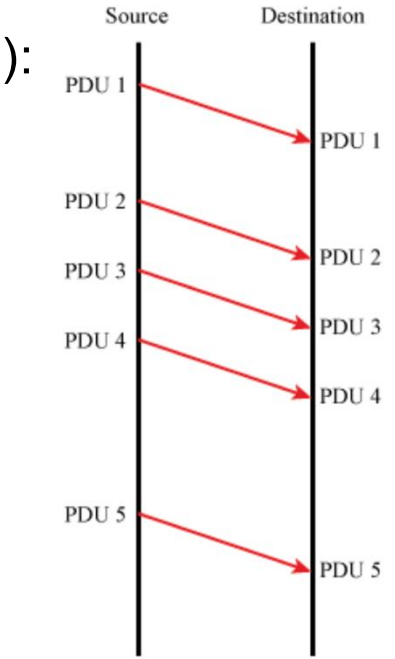
Protocol Control Information (PCI):

- Sequence Bits
- Check Bits

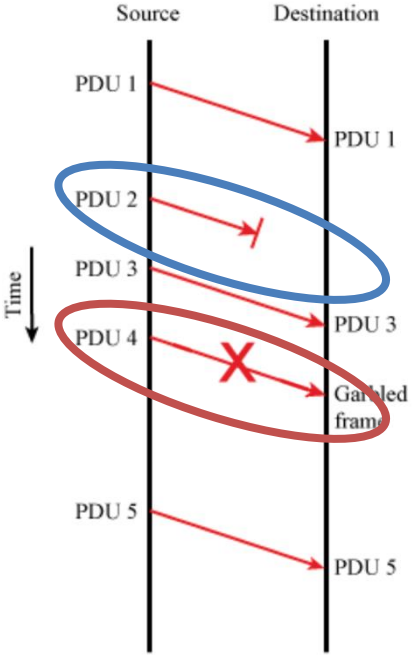
Vertical-time sequence diagram

Types of errors:

- Lost PDU
- Damaged PDU



Error-Free Transmission



Transmission with Errors

Automatic Repeat Request (ARQ) Process

University of Stavanger

Ingredients of ARQ:

- **Acknowledgement (ACK)** sent by the receiver of the correct delivery
- **Timeout** within which the sender waits for the ACK
- **Retransmission**, if ACK does not arrive before the timeout
- PDUs are **stored** at the sender
- Buffer/**windows** of sender and receiver





Stop-and-wait ARQ

1. The transmitter (Tx) sends one PDU at a time
2. The Tx waits for the ACK
3. The Receiver (Rx) checks the errors and order
4. If the Rx detect an error, it sends a Negative ACK (NAK)
5. If the timeout expires, the Tx sends again the PDU

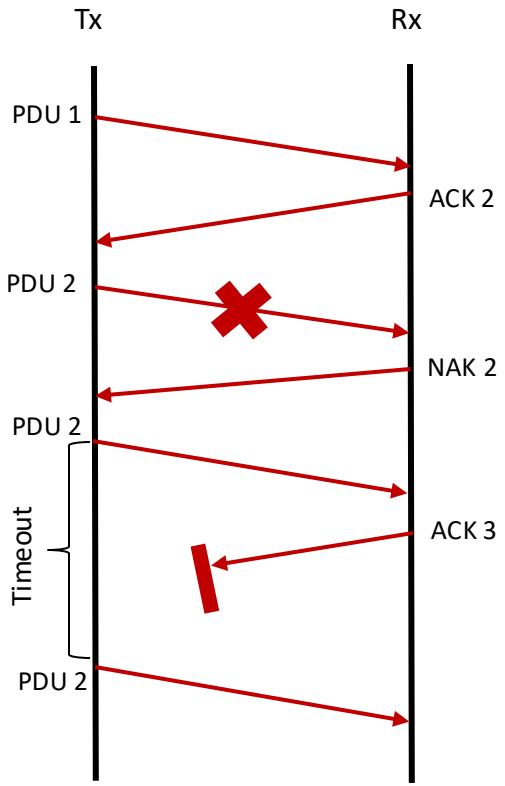
Why the NAK?

Which is the value of the timeout?

A bit higher than the Round-Trip Time (RRT).

Critical!

1. *Estimation of the RTT*
2. *Waiting the RTT (and turnaround time) to send the the next PDU*



Go-back-N ARQ

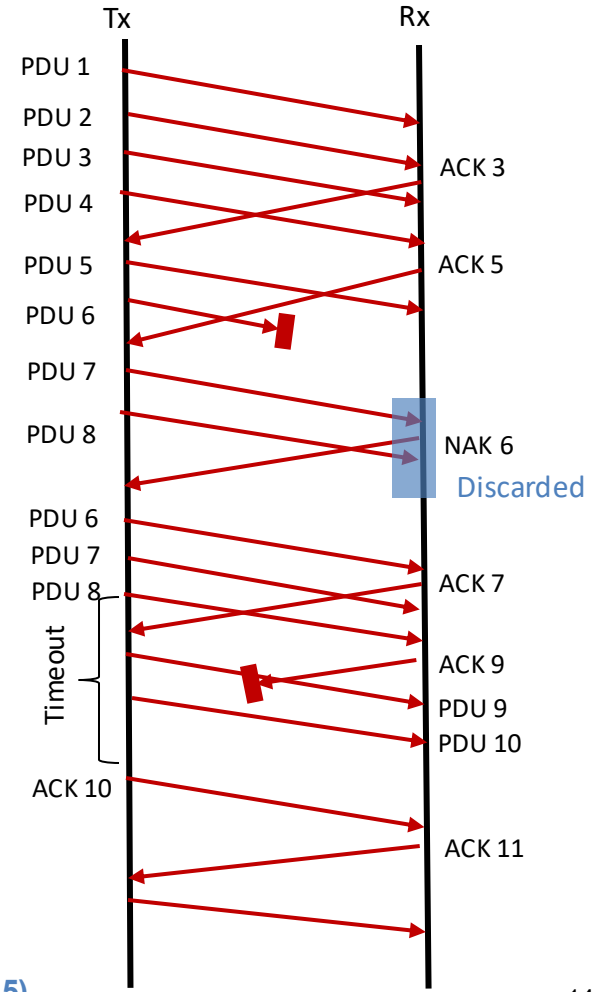
- Tx window = *W*; Rx window= 1

1. The Tx can send *W* PDUs without receiving ACKs
2. The Rx sends a (cumulative) ACK for the last correct and in-order PDU
3. If the timeout expires, the Tx sends an ACK with the expected sequence number

Compared to stop-and-wait ACK?

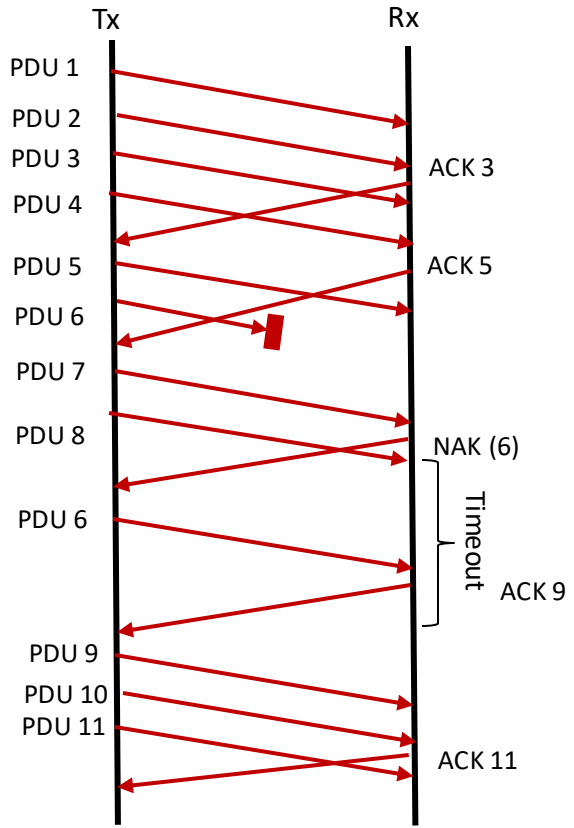
More efficient because, not waiting for the reception of the ACK for each PDU, the connection is still **utilized**.

More and more efficient?



Selective Repeat

- The Tx behaves as go-back-N ARQ
- The Rx cand send (selective) NAK on a single PDU
- The RX accepts **not-in-order** PDUs and send the related ACK
- The Rx window is higher than 1



Other details

University of Stavanger

- And if the information flows are bidirectional?
  - **Piggybacking**: ACK in the PDU

- Can the sequence number become infinite?
  - **Modulo K**

- Which is the value for K?
  - **Depends** on the kind of ARQ (and the related windows)
    - Stop-and-wait ARQ: 2
    - Go-back-N: W+1
    - Selective repeat: 2W+1

Limit of ARQ in Wireless Applications

University of Stavanger

- In a wireless link, **BER** can be *quite high*, which would result to large number of retransmissions

- In some cases (see satellite), **propagation delay** is *very long* w.r.t. transmission time of a frame

Need of Error Correction Codes!

University of Stavanger

Error Correction

University of Stavanger

Error Correction

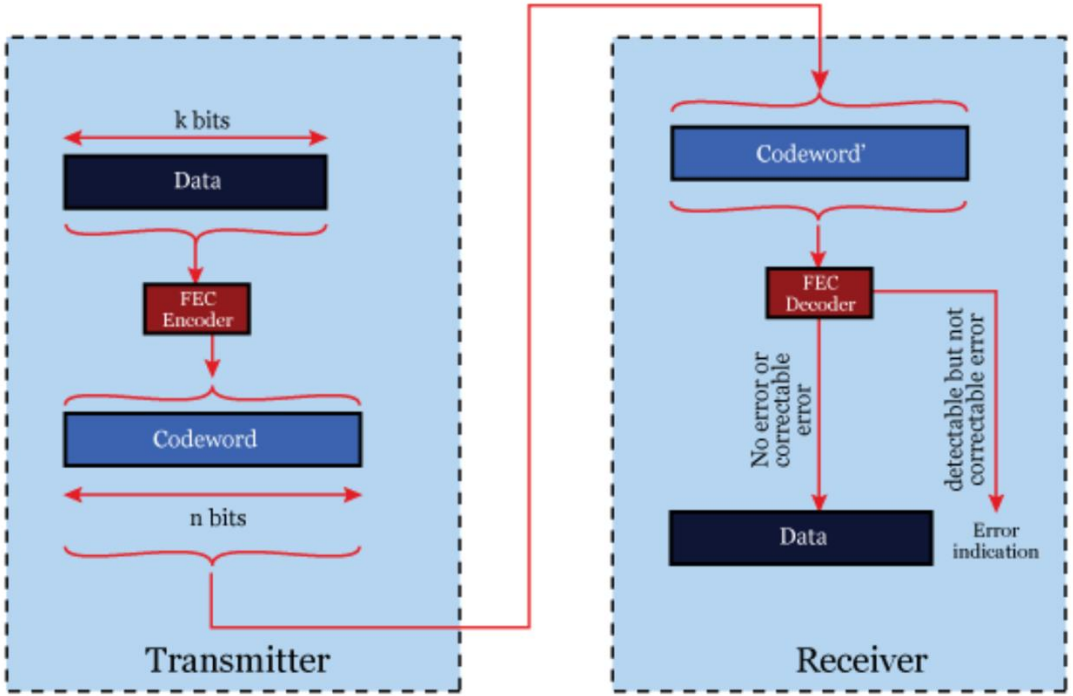
- Enable the receiver to correct errors

- *k*-bit block of data
- *n*-bit **codeword** by FEC encoder
- *n* > *k*, adding **redundancy**

- *Example: 0 → 0000, 1 → 1111 (k=1, n=4)*
  If 0010 received, which is the transmitted?
  If 0011?

Types of FEC:
- Block Codes
- Convolutional Codes

Error Correction



FEC decoder outcomes:

1. Original data block, if no bit errors
2. Original data block, if possible to detect and correct errors
3. Uncorrectable error, if possible to detect but not correct errors
4. Not original block, if detect error but not correct it properly (*rare*)
5. Not original block, if not detect error (*more rare*)

University of Stavanger

Block Codes

- **Hamming distance** is the number of bits in which two *n*-bit binary sequences disagree
  - *Example*: $\mathbf{v}_1 = 011011$ and $\mathbf{v}_2 = 110010$
    $d(\mathbf{v}_1, \mathbf{v}_2) = 3$

- **Design of a block code**: design of a function of the form $\mathbf{v}_c = f(\mathbf{v}_d)$
  - $\mathbf{v}_d$: vector of data block
  - $\mathbf{v}_c$: vector of codeword

- (n, k) block code: 2^k valid codewords out of 2^n
  - **Redundancy**: $(n-k)/k$
  - **Code Rate**: k/n

Block Codes

- **Minimum Hamming distance** of a code: $d_{\min} = \min_{i \neq j}[d(\boldsymbol{w}_i, \boldsymbol{w}_j)]$
  - $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_s$ are $s = 2^k$ codewords
  - *Example*:

Data Block	Codeword
00	c_1=00000
01	c_2=00111
10	c_3=11100
11	c_4=11011

$d_{\min} = 3$

- Number of guaranteed **correctable** errors: $\left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$
- Number of guaranteed **detectable** errors: d_{min}-1

Design of a block code

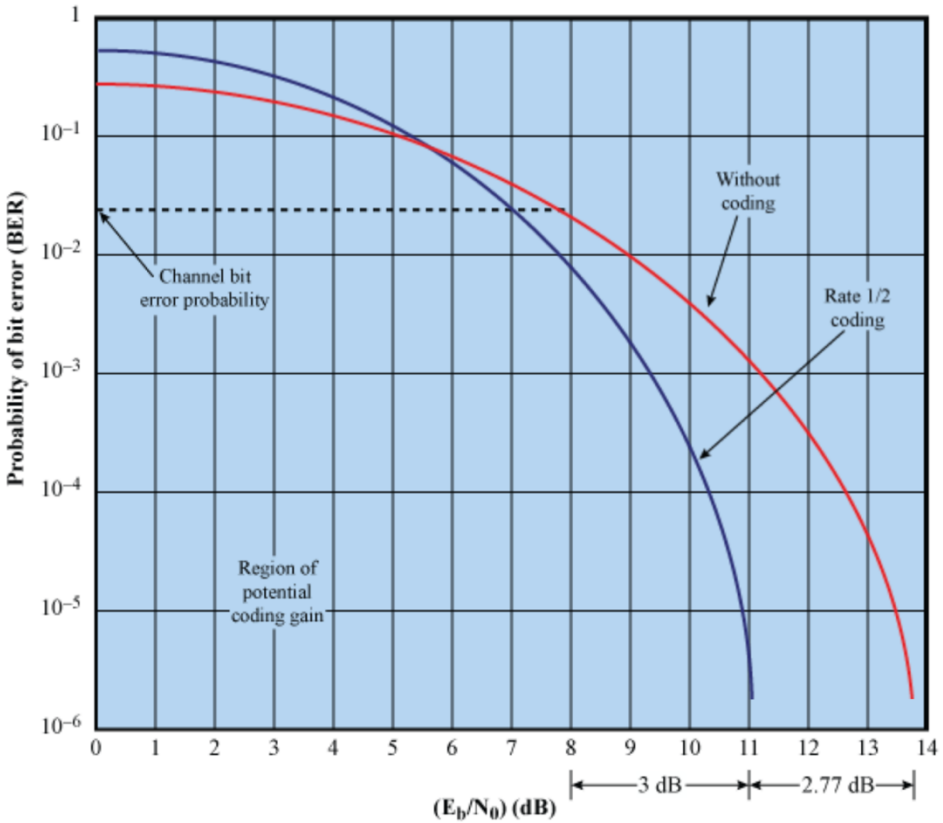
University of Stavanger

Considerations:

1. Given n and k, d_{min} largest possible
2. Encoding and decoding relatively easy (memory and processing)
3. $(n - k)$ small to reduce bandwidth
4. $(n - k)$ large to reduce error rate

Find a trade-off!

Improvement of Coding



Coding Gain [*dB*]:
reduction of E_b/N_0 *needed to obtain a certain BER*

Hamming Code

Block length	$n = 2^m-1$
Number of data bits	$k = 2^m-m-1$
Number of check bits	$m = n-k$
Minimum distance	$d_{min} = 3$

- Rarely used
- Correct single-bit errors
- Same structure as error detection logic
  - Preserve *k* data bits and add (*n-k*) check bits

Hamming Code

- **Hamming check bits**
  - Inserted at positions at power of 2 (starting from right)
  - *Calculation*: XOR of the position number of the data-block bits that are equal to 1

- **Syndrome word**
  - Calculation: XOR of the position number of the received data-block bits that are equal to 1 and the Hamming check bits
  - Characteristics:
    - All 0s, no errors
    - One 1, error in the check bits
    - More that one 1, indication of the error position

Hamming Code

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	0	0	1	1		1	0	0		1		
Check bit					0				1		1	1
Trans. Block	0	0	**1**	**1**	**0**	**1**	0	0	**1**	**1**	**1**	**1**

Calculating the check bits:

D3	0011
D7	0111
D9	1001
D10	1010
XOR (D3, D7, D9, D10)	**0111**

What if we receive **1011010011<u>1</u>11** ?

Calculating the syndrome word:

D3	0011
D7	0111
D9	1001
D10	1010
D12	1100
Check bits	**0111**
XOR	**1100**

Position of the error

Cyclic Codes

- The cyclic shift a valid code of one place to the right is also a valid code
  - $\mathbf{c}=(c_0, c_1, \ldots, c_{n-1})$ is valid, then $(c_{n-1}, c_0, c_1, \ldots, c_{n-2})$ is also valid

- Examples of cyclic codes:
  - Bose-Chaudhuri-Hocquenghem (BCH) code *Most powerful cyclic code* *Widely used in wireless*
  - Reed-Solomon (RS code) *Subclass of BCH codes* *Well suited for burst error correction*

Parity-Check Matrix Codes

- Parity-check bit

  $c_1 \oplus c_2 \oplus \ldots \oplus c_n = 0$

  – Even parity

  – $\oplus$: modulo-2 addition or XOR

- Parity-check matrix

  $h_{11} c_1 \oplus h_{12} c_2 \oplus \ldots \oplus h_{1n} c_n = 0$

  $h_{21} c_1 \oplus h_{22} c_2 \oplus \ldots \oplus h_{2n} c_n = 0$

  .

  .

  .

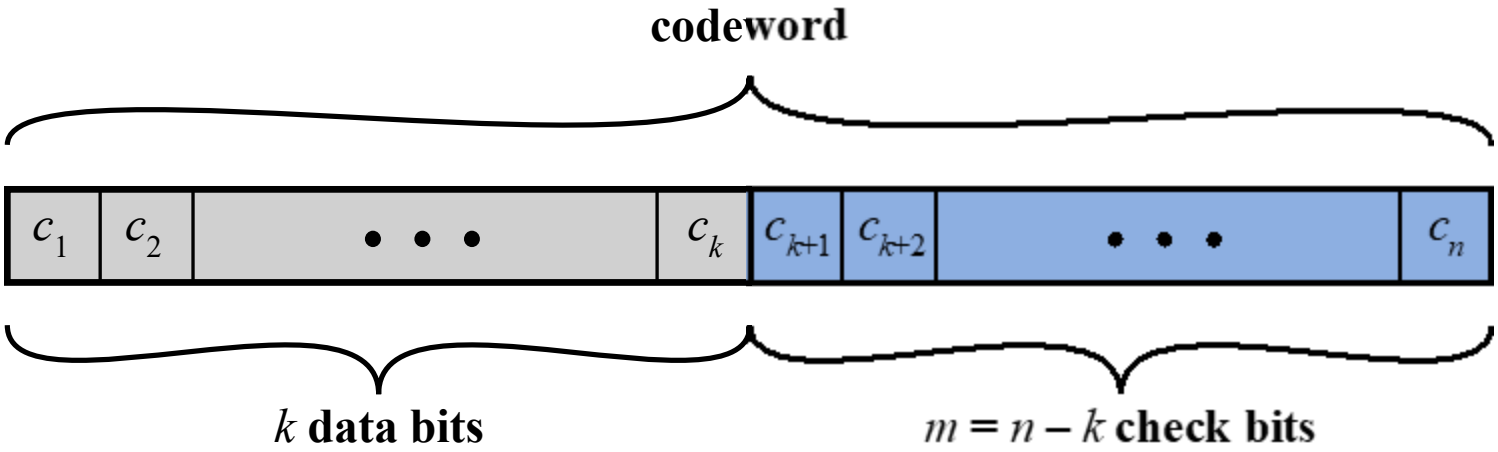
  $h_{m1} c_1 \oplus h_{m2} c_2 \oplus \ldots \oplus h_{mn} c_n = 0$

  – $\mathbf{H} = [h_{ij}]$ is a $m \times n$ matrix with $h_{ij} = \{0,1\}$

  – $Hc^T = cH^T = 0$

Parity-Check Matrix Codes: Codeword

- Systemic code



Property of the Parity-Check Matrix

- $(n - k)$ linear independent equations

- $\mathbf{H} = [\mathbf{A}\ \mathbf{I}_{n-k}]$
  - $\mathbf{I}_{n-k}$: $(n - k) \times (n - k)$ identity matrix
  - $\mathbf{A}$: $(n - k) \times k$ matrix
  - Linear independence if determinant of $\mathbf{A}$ is not zero

- Given the 2^k possible sets of data bits, it is possible to uniquely determine the $(n - k)$ check bits

Parity-Check Matrix Codes: Encoding

- Example: (7, 4) check code

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$c_5 = c_1 \oplus c_2 \oplus c_3$$
$$c_6 = c_1 \oplus c_3 \oplus c_4$$
$$c_7 = c_1 \oplus c_2 \oplus c_4$$

- $k \times n$ generator matrix c=uG
  - $\mathbf{u} = [u_i]$: data bits
  - $\mathbf{G} = [\mathbf{I}_k \mathbf{A}^T]$

Parity-Check Matrix Codes: Encoding

Data bits				Check bits		
c_1	c_2	c_3	c_4	c_5	c_6	c_7
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	1	1

Parity-Check Matrix Codes

- **Error detection**, if $\mathbf{Hc}^T$ yields a nonzero vector
  - The resulting vector is called syndrome
  - The syndrome indicates which parity-check equations are not satisfied

- **Error correction**: choose the valid codeword that is closest to the nonvalid codeword in term of Hamming distance
  - Brute-force approach, but not feasible for large data blocks
  - In this case, various alternative approaches have been developed

Low-Density Parity-Check (LDPC) Codes

- A **regular** LDPC code has H with these properties:
  1. Each code bit is involved with w_c parity constraints, and each parity constraint involves w_r bits.
  2. Each row of H contains w_r 1s, where w_r is constant for every row.
  3. Each column of H contains w_c 1s, where w_c is constant for every column.
  4. The number of 1s in common between any two columns is zero or one.
  5. Both w_r and w_c are small compared to the number of columns and then number of rows ($w_r << n$ and $w_c << m$)
- Property 5 means **low density**.

LDPC Codes

- Since property 2 and 3 are usually violated to avoid linear dependency, there are **irregular** LDPC codes
  - Where the average number of 1s per row and the average number of 1s per column are small compared to the number of columns and the number of rows

LDPC Codes

University of Stavanger

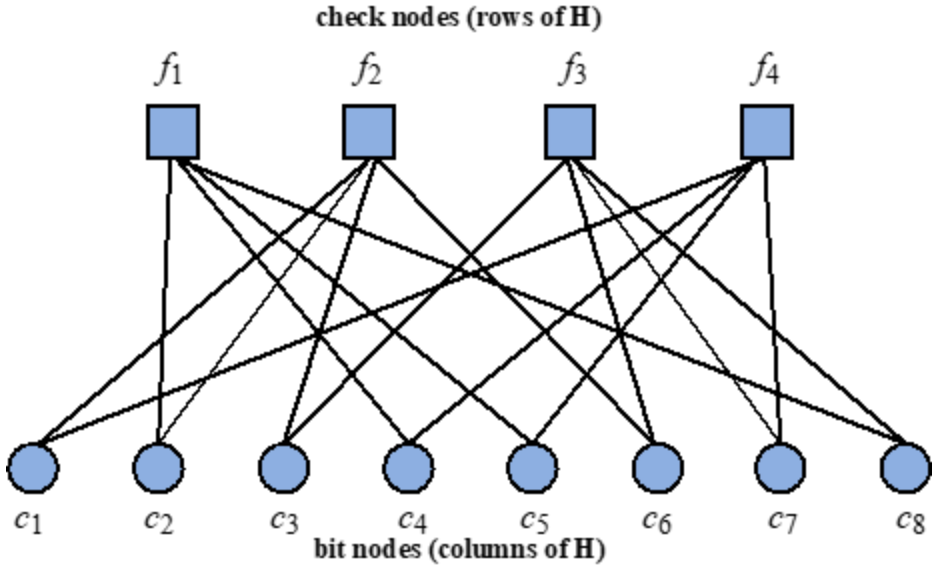
- **Code construction**: how to determine **H**
  - Various approaches have been developed
    - Gallagher's and MacKay and Neal's ones
    - Nonsystematic codes, difficult to encode
    - Repeat-accumulate code, which converts to a systematic one

- **Encoding**:
  - Easy for repeat-accumulate codes
  - For the other codes, resource-intensive operation
    - Often involve the Tanner graph

LDPC Codes

- The **error detection** is simple as for generic parity-check codes
- The **error correction** is complex
  - Many approaches use a **Tanner graph**

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



LDPC Codes

- Iterative decoding procedure (message passing):
  1. Each bit node sends its bit value to the linked check nodes.
  2. Each check node calculates a bit value for each linked bit node based on its corresponding equation (XOR of the other bit values) and sends the values to the bit nodes.
  3. Each incoming values to each bit nodes is an estimate of the corrected values (together with the original value). The final estimate is the majority value.

- The procedure is repeated until all the constrain equations are satisfied

- A probabilistic approach can be also used

LDPC Codes

University of Stavanger

Constraint equations

Row of H	Check node	Equation
1	f_1	$c_2 \oplus c_4 \oplus c_5 \oplus c_8 = 0$
2	f_2	$c_1 \oplus c_2 \oplus c_3 \oplus c_6 = 0$
3	f_3	$c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$
4	f_4	$c_1 \oplus c_4 \oplus c_5 \oplus c_7 = 0$

Check node	Messages				
f_1	received:	$c_2 \to 1$	$c_4 \to 1$	$c_5 \to 0$	$c_8 \to 1$
	sent:	$0 \to c_2$	$0 \to c_4$	$1 \to c_5$	$0 \to c_8$
f_2	received:	$c_1 \to 1$	$c_2 \to 1$	$c_3 \to 0$	$c_6 \to 1$
	sent:	$0 \to c_1$	$0 \to c_2$	$1 \to c_3$	$0 \to c_6$
f_3	received:	$c_3 \to 0$	$c_6 \to 1$	$c_7 \to 0$	$c_8 \to 1$
	sent:	$0 \to c_3$	$1 \to c_6$	$0 \to c_7$	$1 \to c_8$
f_4	received:	$c_1 \to 1$	$c_4 \to 1$	$c_5 \to 0$	$c_7 \to 0$
	sent:	$1 \to c_1$	$1 \to c_4$	$0 \to c_5$	$0 \to c_7$

Messages sent and received (steps 1. and 2.)

Bit node	Codeword bit	Messages from check nodes	Decision
c_1	1	$f_2 \to 0$ $f_4 \to 1$	1
c_2	1	$f_1 \to 0$ $f_2 \to 0$	0
c_3	0	$f_2 \to 1$ $f_3 \to 0$	0
c_4	1	$f_1 \to 0$ $f_4 \to 1$	1
c_5	0	$f_1 \to 1$ $f_4 \to 0$	0
c_6	1	$f_2 \to 0$ $f_3 \to 1$	1
c_7	0	$f_3 \to 0$ $f_4 \to 0$	0
c_8	1	$f_1 \to 0$ $f_3 \to 1$	1

Estimation of the codeword bit values (step 3.)

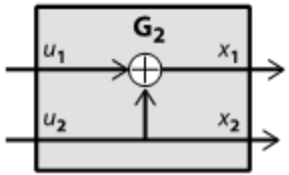
Polar Codes

- Perform worse than LDPC codes, but efficient implementation
- Code design is based on the concept of a synthetic channel
- **Recursive** FEC technique
  - Repeated use of XOR
  - Codeword length that is power of 2

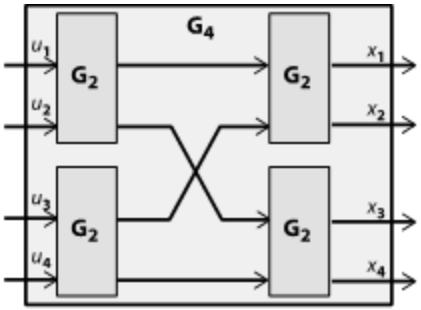
Example:

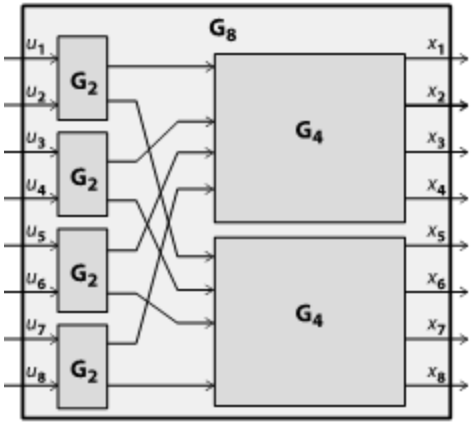
$x_1 = u_1 \oplus u_2$

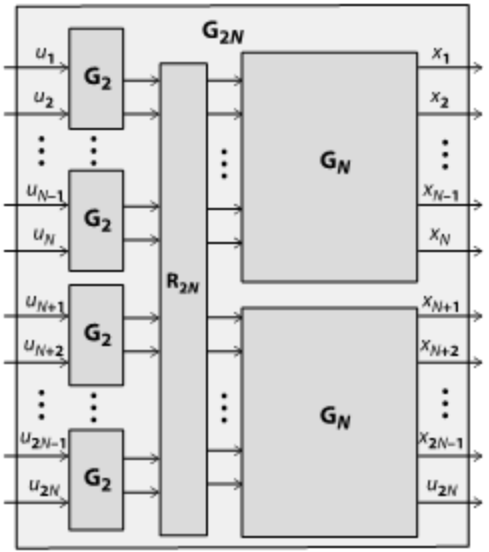
$x_2 = u_2$



Polar Codes







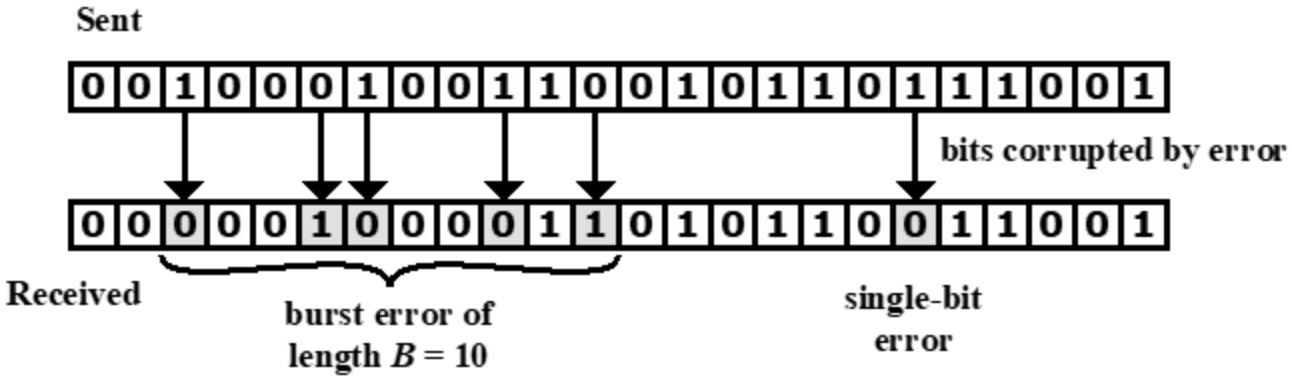
- R_{2N} is a permutation called **reverse shuffle**
  - Copies its odd-numbered inputs to its first N outputs and copies its even-numbered inputs to its last N outputs
- For a (N, K) code, K data bits are each assigned to one of the input lines and the remaining lines are frozen (fixed value of 0)

Polar Codes

University of Stavanger

- The efficient **decoding** is a challenge
  - One possible approach: successive cancelation
  - There are more efficient but more complex approaches

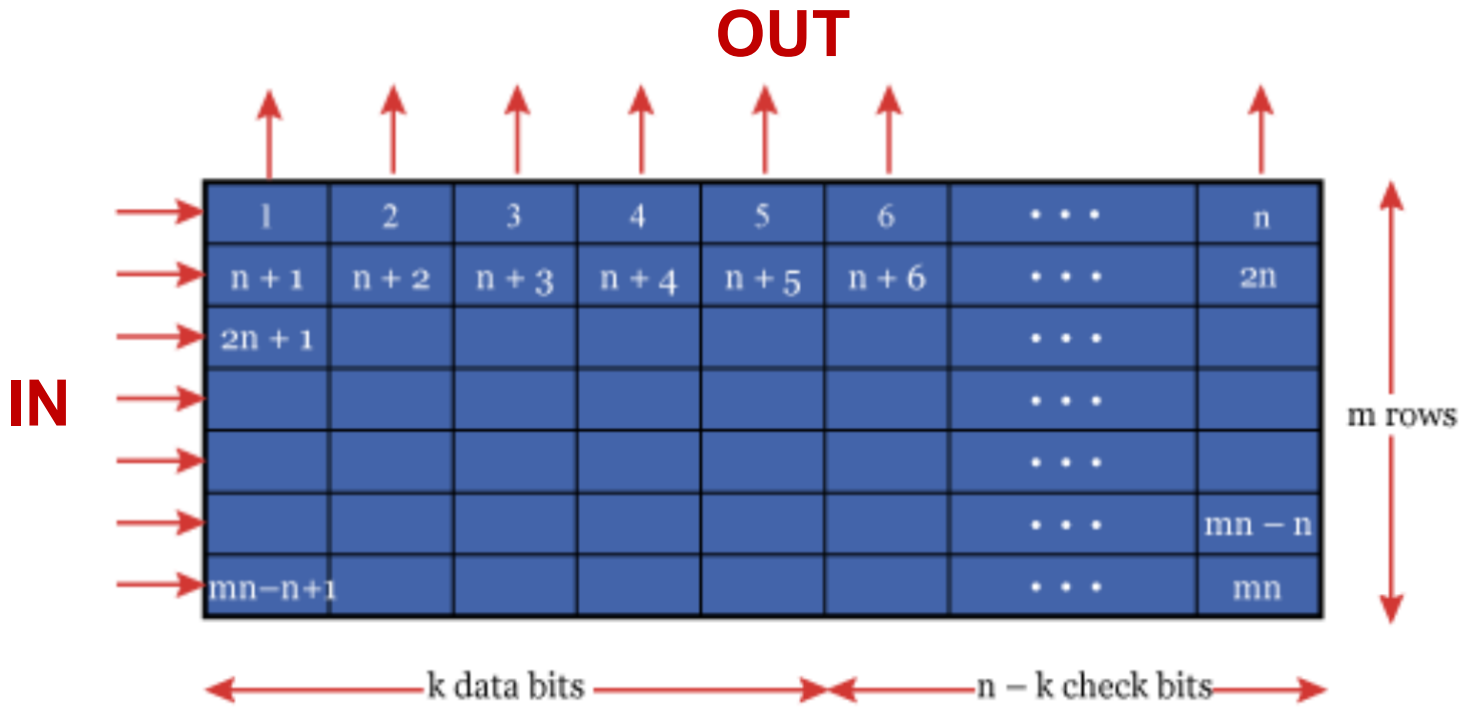
Error burst



- Caused by impulse noise or by fading
- Greater at higher data rates

Block Interleaving

For reducing burst error effect, spread out over a number of separated blocks



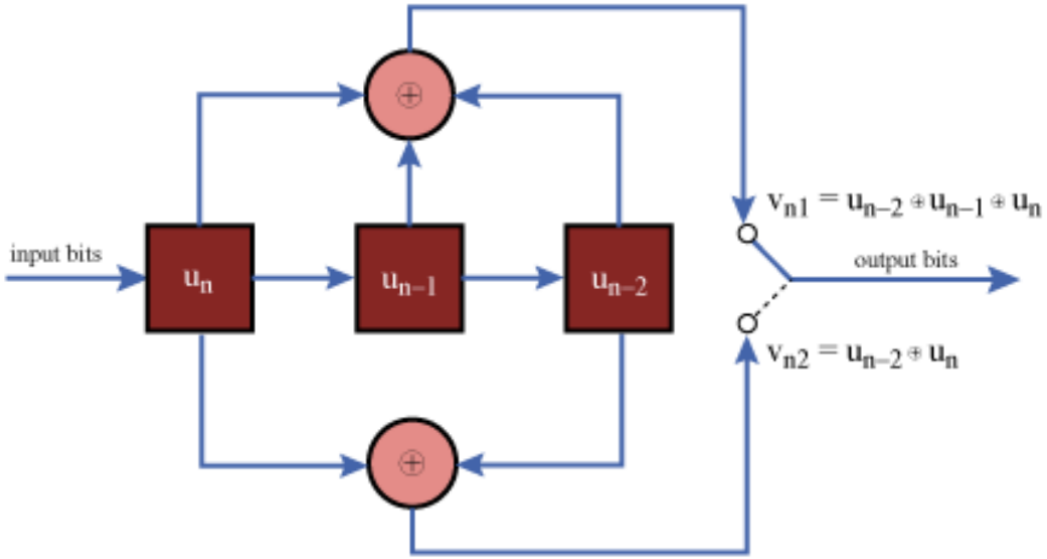
University of Stavanger

Convolutional Codes

- **Blocks codes** (especially for large *n*) are **not convenient** when the data stream is more or less continuous
- Convolutional codes generates redundant bits **continuously**
- Parameters of convolutional codes (*n*, *k*, *K*)
  - *k*: data bits processed at a time
  - *n*: output bits from each *k* bits
  - *K*: constraint factor (*K*-1 = number of previous data sets)
- *n* output bits are depending on $K \times k$ input bits

Convolutional Codes - Encoder

- Encoder shift register

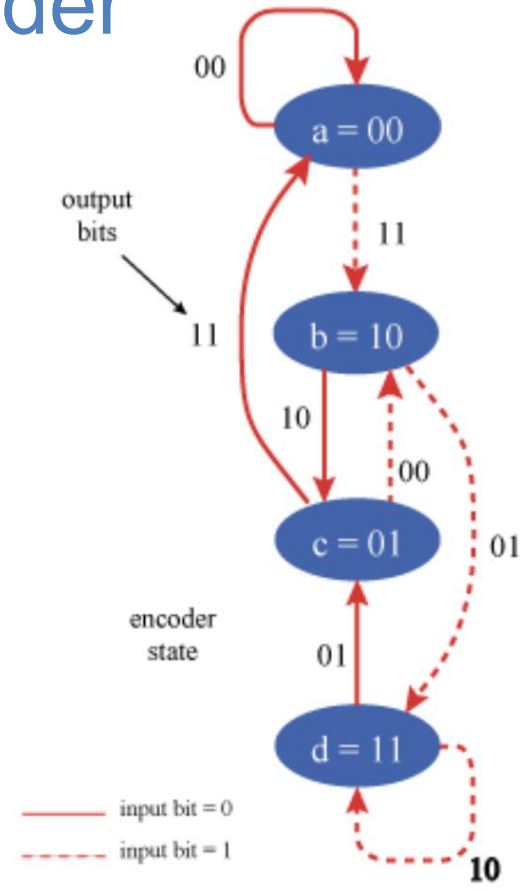


Which are the values of (n, k, K)?

(2, 1, 3)

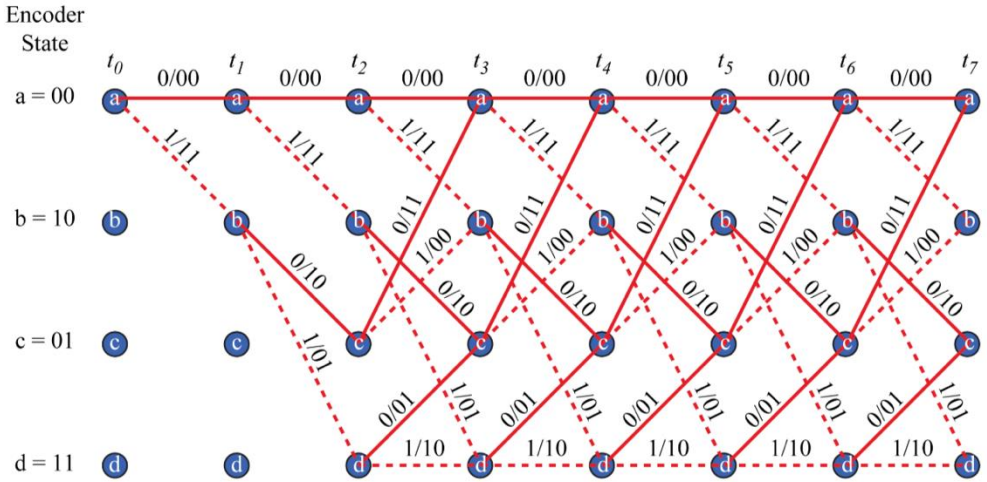
Convolutional Codes - Encoder

- Encoder state diagram
  - State = K-1 past values



Convolutional Codes - Decoder

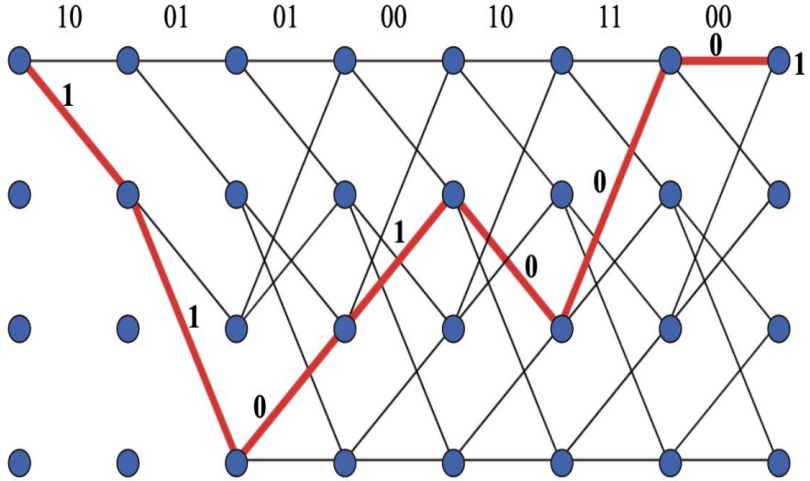
- Trellis diagram
  - state diagram over time
  - Example
    - Path: a-b-c-b-d-c-a-a
    - Output:
      11 10 00 01 01 11 00
    - Input: 1011000
  - Invalid path: a-c



Convolutional Codes - Decoder

- Viterbi algorithm
  - Find most likely input that produced the invalid output
  - Metric: Hamming distance
  - Example:
    - w = 1001010010110
    - Decoding window b = 7
    - Animation:

https://media.pearsoncmg.com/ph/esm/ecs_stallingsbeard_wcns_1/animations/10_11_trellis_diagram_for_encoder/index.html

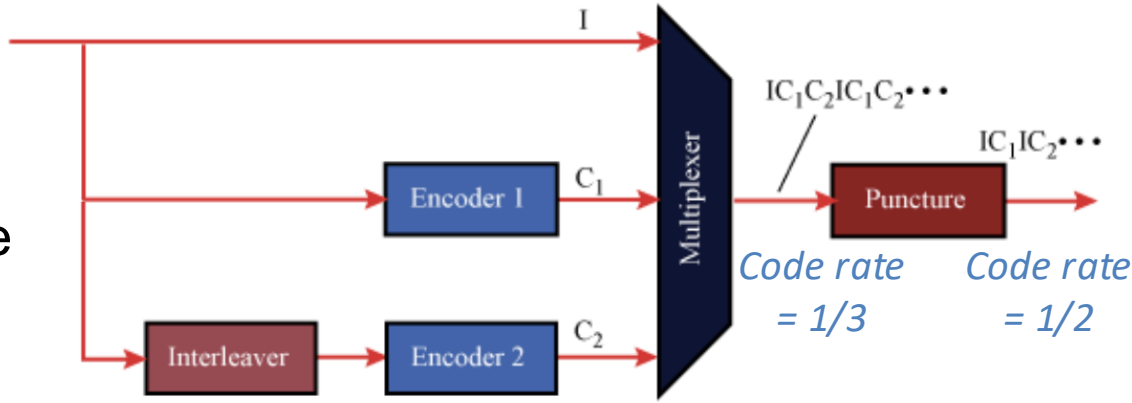


Turbo Codes

- For higher and higher speeds
- BER performance are close to the Shannon limit
- Based on convolutional encoding
- Used in 3G and 4G

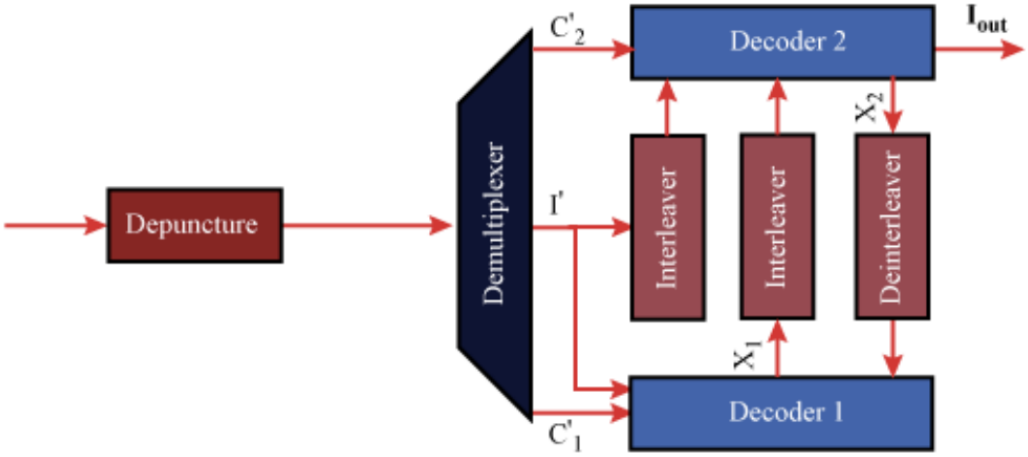
Turbo Codes - Encoder

- Puncturing = Process to get only one check bits
- Input bit preserved
- Each encoder produce a single check bit
- Recursive systematic convolutional (RSC) code



Turbo Codes - Decoder

- Depuncturing= estimating missing check bit
- Soft decision decoding
- Several iterations
  - High level of confidance
  - delay



University of Stavanger

Hybrid ARQ

Hybrid ARQ (HARQ)

- *Good channel*, FEC may add unnecessary redundancy 
- *Bad channel*, ARQ may cause excessive delay 

- **Type-I HARQ**:
  - Use a combination of FEC to correct common errors and ARQ for retransmission when FEC cannot make corrections

- **Type-II HARQ**:
  - Alternate messages with only error detection and messages with FEC

University of Stavanger

Additional Approaches

- Soft decision decoding
  - Level of confidence
- Chase combining
  - Not discard previous not-corrected frames
- Incremental redundancy
  - Increment redundancy with retransmission
- Puncturing
  - Remove bits, instead of changing FEC

Learning Material

• Error Detection	5.6, 10.1	14.3
• Automatic Repeat Request	5.6, 10.4	
• Error Correction	5.6	
– Hamming Codes	10.2	14.2
– Cyclic Codes	10.2	
– Low-Density Parity Codes	10.2	14.3, 14.4, 14.6
– Polar Codes		14.5, 14.6
– Convolutional Codes	10.3	
– Turbo Codes	10.3	
• HARQ	10.4	14.7