

# Digital Data Communication Techniques

Dr Steve Gordon  
ICT, SIIT

# Contents

- Asynchronous and Synchronous Transmission
- Handling Errors
  - Types of Errors
  - Error Detection
  - Error Correction



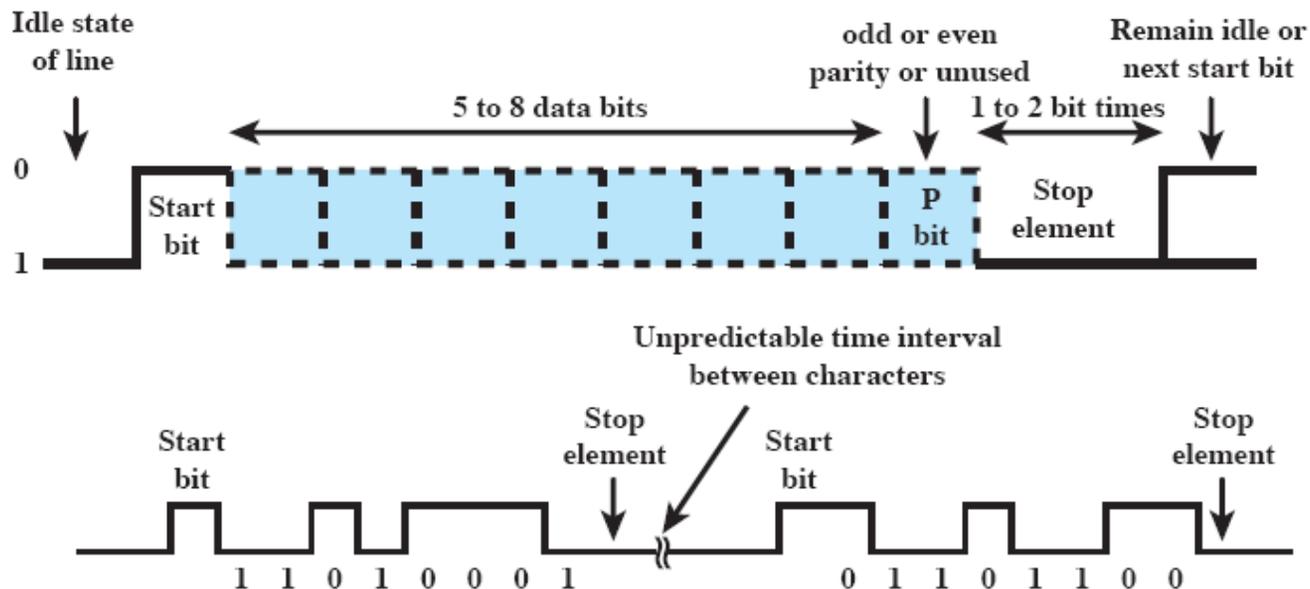
# Asynchronous and Synchronous Transmission

- Problem:
  - Transmitter sends a sequence of bits
  - Receiver must know when a bit starts and finishes
  - Sender and receiver must be synchronized
    - If not, then receiver may sample at wrong time and get bit errors
- Two solutions to synchronizing clocks
  - Asynchronous transmission
  - Synchronous transmission



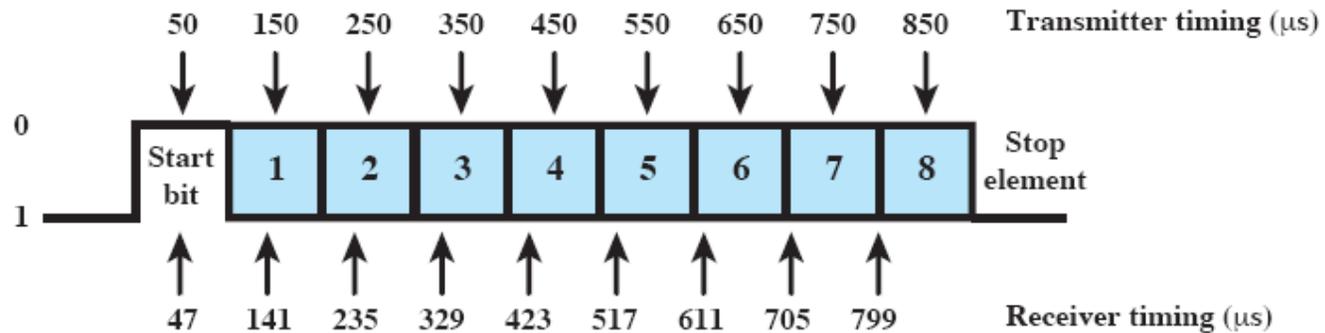
# Asynchronous Transmission

- Send a character at a time (characters is 5 to 8 bits)
  - A special bit is used to indicate the start of character and end of character
  - Example:
    - When no character being transmitted, binary 1 is transmitted
    - Start bit is binary 0; then character bits are sent (5 or 8 bits); then stop bit (binary 1)



# Asynchronous Transmission

- Can tolerate modest timing errors
  - Example: 10kb/s data rate; receiver is fast by 5% can still correctly sample data
    - If receiver fast by 6% (figure below), then error

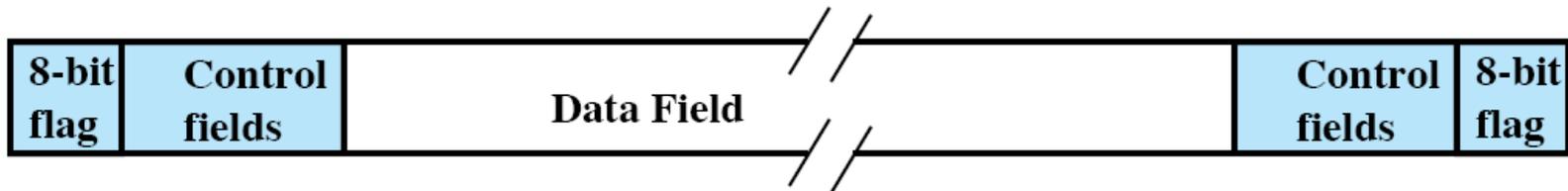


- Asynchronous transmission is simple and cheap
  - Requires overhead of two to three bits per character
  - Good for data with large gaps, e.g. keyboard data



# Synchronous Transmission

- Block of data transmitted sent as a frame
- Clocks at transmitter and receiver must be synchronized
  - Can use separate clock line
  - Or embed clock signal in data
- Need to indicate start and end of block
  - use pre-amble and post-amble bit patterns
- More efficient (lower overhead) than asynchronous transmission



- Example: HDLC (covered later) contains 48-bits of pre/post-amble. With 1000 bytes of data, overhead is 0.6% (compared to about 20% with asynchronous)



# Types of Error

- An error occurs when a bit is altered between transmission and reception
  - E.g. transmitter sends a 1 but receiver thinks it is a 0
- Single bit errors
  - Only one bit altered
  - Caused by noise, the SNR is too low for receiver to determine the correct bit
- Burst errors
  - Contiguous sequence of  $B$  bits in which first last and any number of intermediate bits in error
  - Caused by impulse noise or by fading in wireless
  - Effect greater at higher data rates

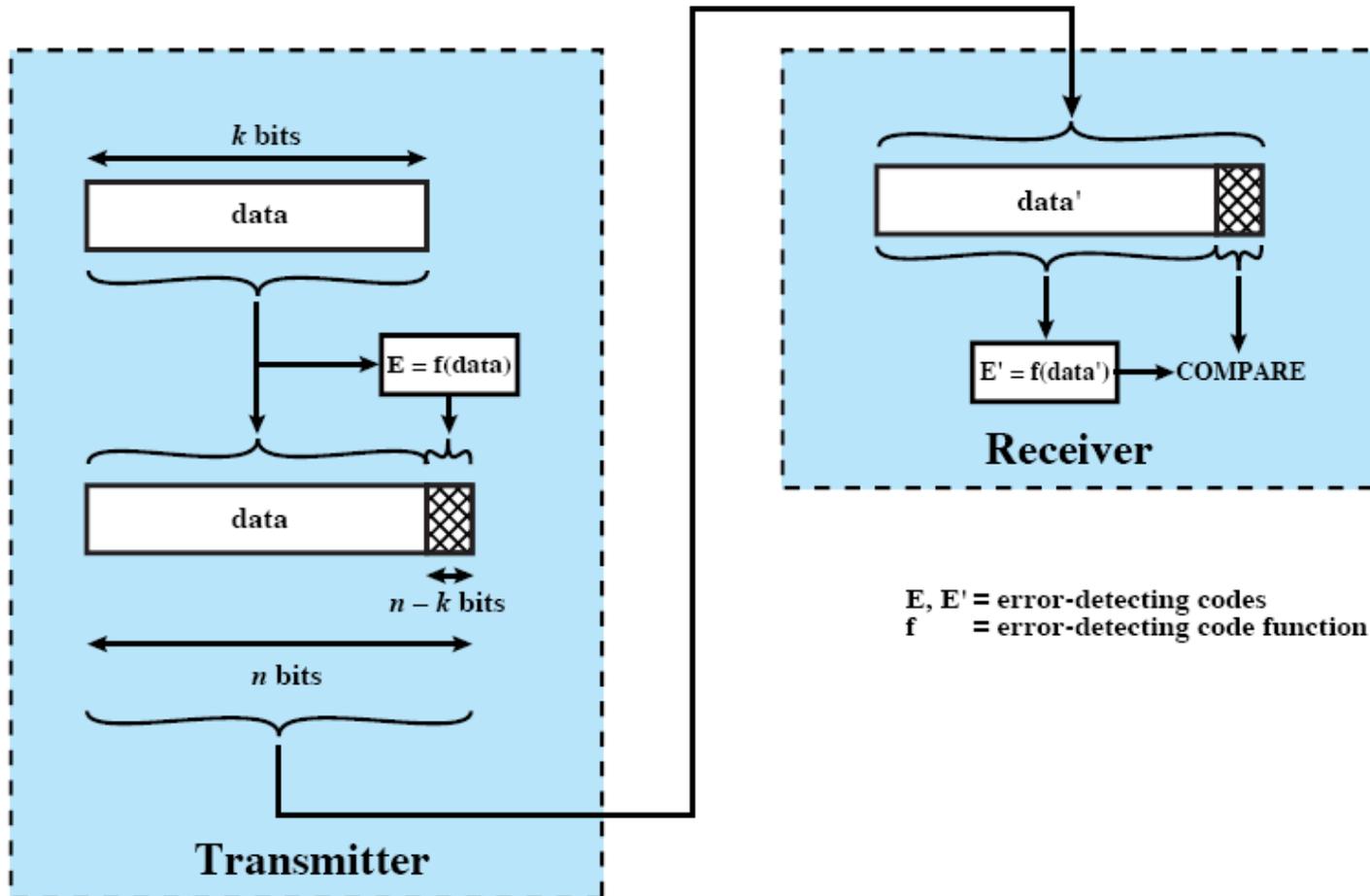


# Error Detection

- We will always have errors
- Detect the errors (so can retry or inform higher layer)
  - Transmitter adds extra information to transmitted data, i.e. an error-detecting code
  - Receiver recalculates the error-detecting code from received data, and compares to received error-detecting code
  - If the same, good. If not, then error (in data or code)
    - Still a chance that an error is not detected
- There are other forms of error detection (covered next topic)
- Simple Error Detecting Code: Parity Check
  - Single parity bit added to character to make the number of 1's even (if using even parity) or odd (if using odd parity)
    - E.g. assume odd parity is used: a 7-bit character 1110001 is sent with an eighth parity bit set to 1. Transmitted: 11110001
    - If 1 bit is in error, receiver will detect it: Receive 11100001
    - If 2 (or even number) bits in error, then not detected



# General Error Detection Process



# Error Detection Example: Parity Check

- Assume a computer wants to send the message “Steve”
- How do we represent this information as digital data?

- Use the ASCII (or similar) character set
- Digital data to send is:

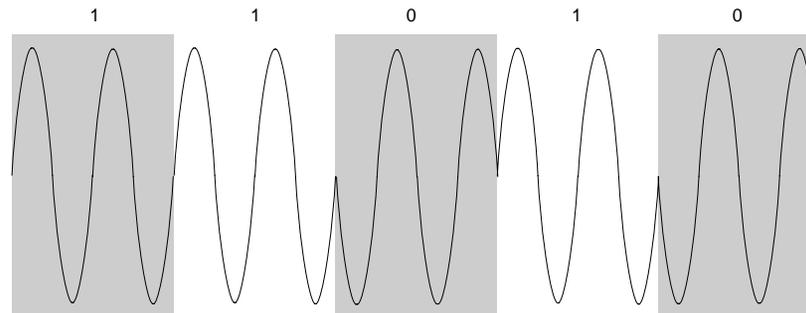
10100111110100110010111101101100101

Char	Dec.	Binary
S	83	1010011
t	116	1110100
e	101	1100101
v	118	1110110

- Example of single bit odd-parity check
  - Take the 7 bits that represent each character
  - Add an extra bit (at the front) to make odd 1's
  - Digital data to send is now:

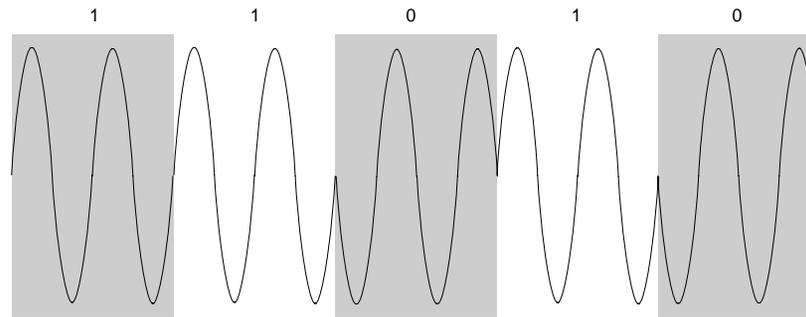
110100111110100111001010111011011100101

- The digital data is now sent, lets assume as an analog signal using BPSK

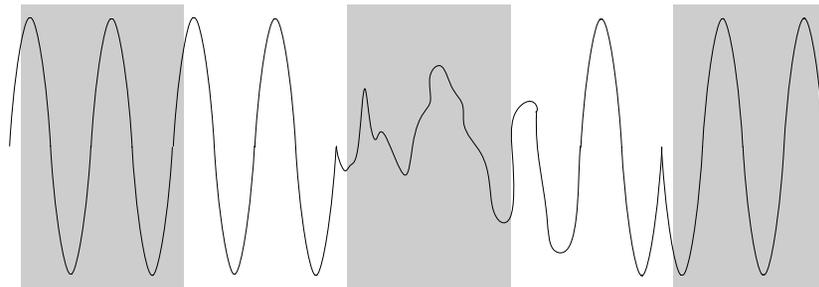


# Error Detection Example: Parity Check

- Transmitted signal (first five bits)



- Received signal (due to transmission impairments)



- The receiver interprets the sequence of bits as:

11100111111010011100101011101011100101



# Error Detection Example: Parity Check

- The receiver error detector looks at each set of 8 bits and checks for errors (remember, using odd parity)
  - First 8 bits: 11110011 – there are six (even number of) 1's
    - ERROR DETECTED!
    - Upon detecting the error, the higher layer may be notified (“The character received is in error – please do something to fix it!”)
    - If there is no error correction functions applied, then the character will not be sent to the higher layer
  - Second 8 bits: 11110100 – there is odd number of 1's
    - No error detected (correct)
  - Third 8 bits: 11100101 - there is odd number of 1's
    - No error detected (correct)
  - Fourth 8 bits: 01111010 – there is odd number of 1's
    - No error detected (INCORRECT!)
  - Fifth 8 bits: 11100101 - there is odd number of 1's
- What does the receiving application/user receive?
  - An error notification, and possibly the characters: “teze”



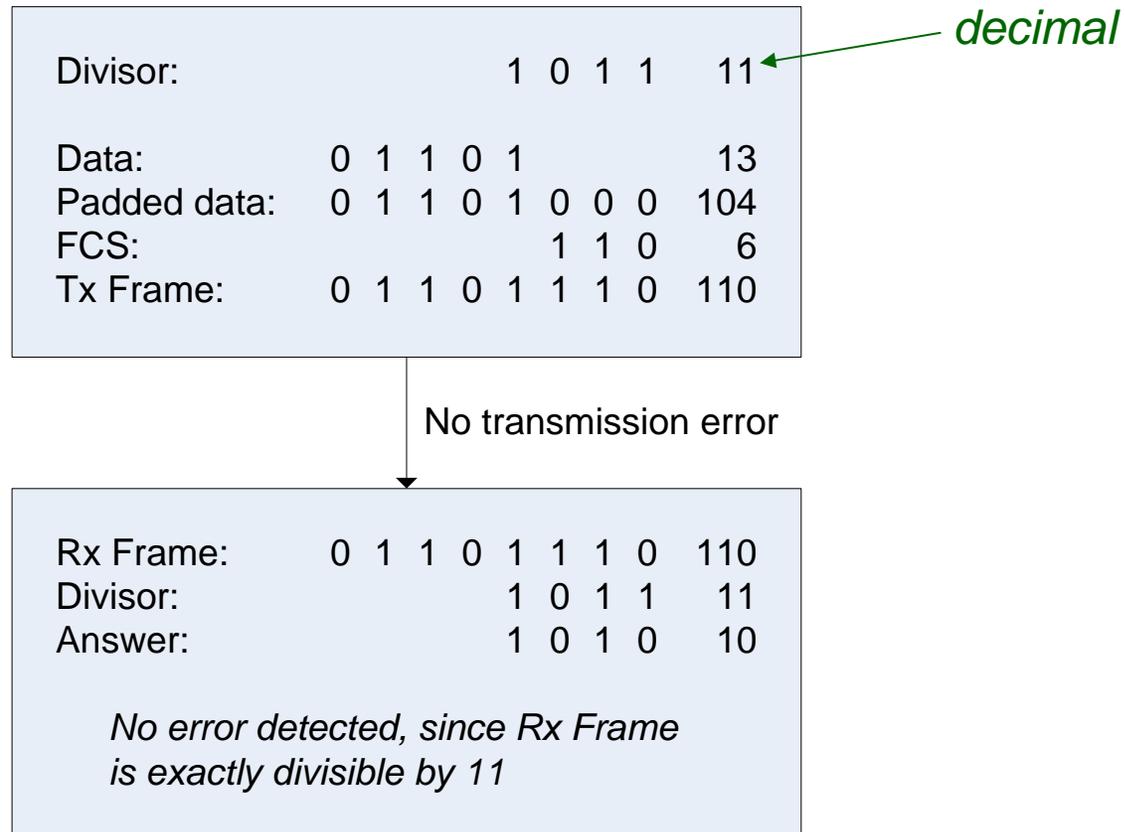
# Cyclic Redundancy Check (CRC)

- One of most common and powerful checks
  - Used as a checksum for data transmission and storage
  - There are different standards, depending on the CRC length and algorithm
- For block of  $k$  bits, transmitter generates an  $(n-k)$  bit frame check sequence (FCS)
- Transmits  $n$  bits which is exactly divisible by some number
  - Divisor is  $n-k+1$  bits in length
- Receiver divides frame by the divisor
  - If no remainder, assume no error
  - If a remainder, then assume error



# Simplified CRC Example

- An example of a simplified algorithm similar to CRC
- Lets assume we have  $k = 5$  bits of data and a  $(n-k) = 3$  bit frame check sequence
- Our divisor is 11. Both transmitter and receiver know this



# Simplified CRC Example

Divisor:	1 0 1 1	11
Data:	0 1 1 0 1	13
Padded data:	0 1 1 0 1 0 0 0	104
FCS:	1 1 0	6
Tx Frame:	0 1 1 0 1 1 1 0	110

2-bit transmission  
error

Rx Frame:	0 1 1 1 1 1 0 0	124
Divisor:	1 0 1 1	11
Answer:	Not an integer	

*Error detected, since Rx Frame is  
NOT exactly divisible by 11*

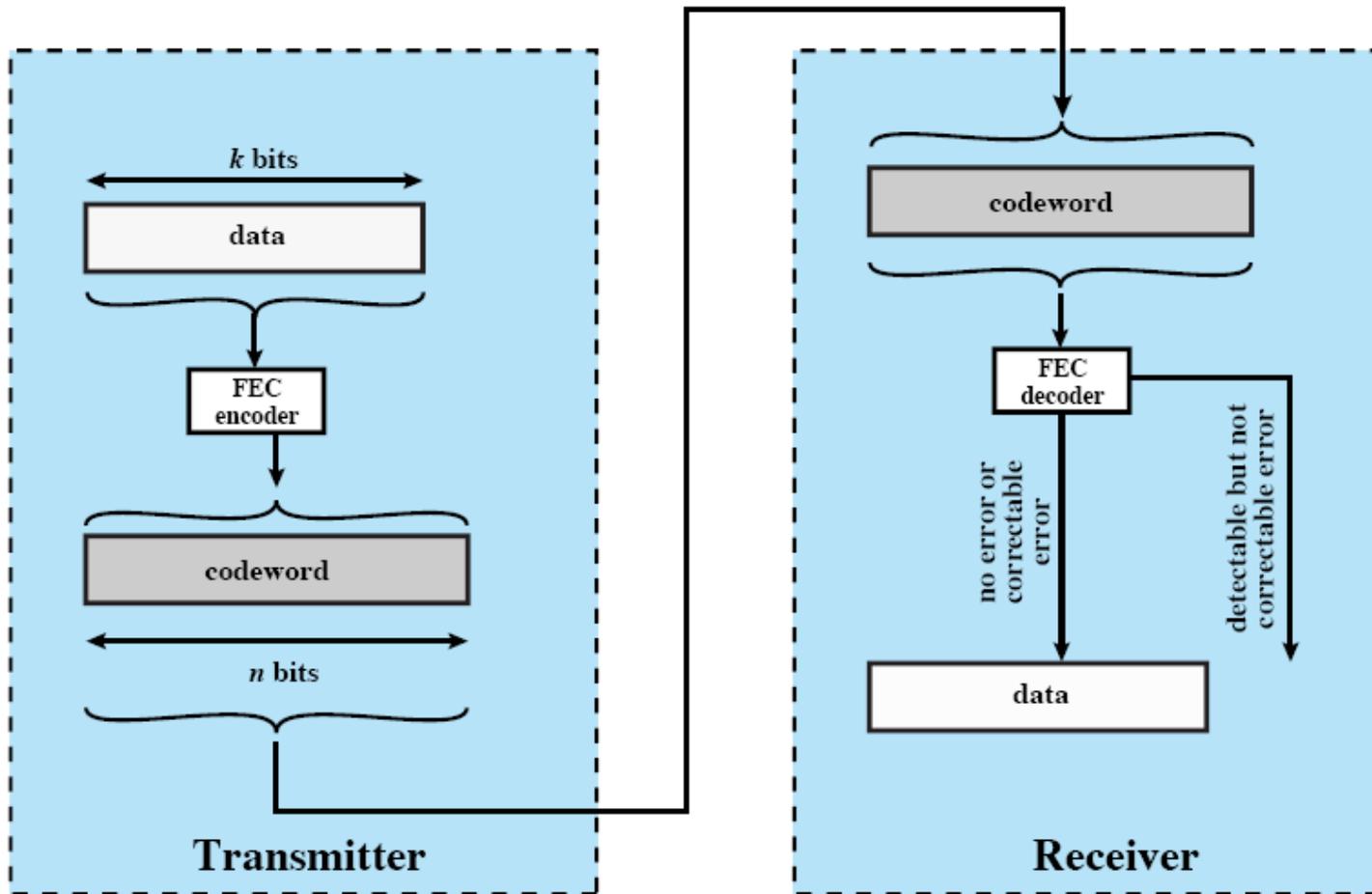


# Error Correction

- Correction of detected errors usually requires data block to be retransmitted (we will see techniques for this in next topic)
- Retransmissions are often not appropriate for wireless applications
  - Bit error rate is high, causing lots of retransmissions
  - When propagation delay long (satellite) compared with frame transmission time, resulting in retransmission of frame in error plus many subsequent frames
- If don't want to retransmit, then instead need to correct errors on basis of bits received
- Forward Error Correction (FEC) provides this
  - “Forward” because the transmitter sends extra data before the error occurs (before = forward of time)



# Error Correction Process



# How Error Correction Works

- Transmitter adds redundancy to transmitted message
- Receiver applies FEC decoder:
  - If no bit errors, input to decoder is same as original codeword, and original data is output
  - Certain error patterns, decoder will detect and correct errors (decoder outputs the original data)
  - Certain error patterns, decoder will detect (but not correct) errors
  - Certain (often rare) error patterns, decoder will not detect nor correct errors (decoder outputs data which is in error)
- Example: block error correction code
  - Map  $k$  bit input onto an  $n$  bit codeword
  - Each codeword is distinctly different
  - If get error assume codeword sent was closest to that received
- Results in reduced effective data rate
  - $(n-k)/n$  is the redundancy of the code;  $k/n$  is the code rate
  - $\frac{1}{2}$  rate code uses double capacity of uncoded system



# Example Error Correcting Code

- Hamming Distance
  - Number of bits of two  $n$ -bit sequences that disagree
    - $v_1 = 011011$        $v_2 = 110001$
    - $d(v_1, v_2) = 3$
- Our FEC encoder maps two bits into 5 bit codeword ( $k=2, n=5$ )

Data	Codeword (C)
00	00000
01	00111
10	11001
11	11110

- If receiver receives an invalid codeword ( $C_r$ ), then assumes the codeword which is minimum Hamming distance from  $C_r$  is the transmitted codeword ( $C_t$ )
  - Only works if only 1 unique codeword with minimum distance









# Performance of Error Detection/Correction

- We want to detect/correct as many errors as possible
  - Because most data applications cannot tolerate errors
    - It is better for a Data Link layer protocol to perform retransmissions over a link to fix errors, then a human user having to retransmit an email
- But both error detection and correction require extra information to be sent over the transmission system
  - If  $k$  bits of useful data, send  $n$  bits of actual data
    - Efficiency of  $k/n$
- Tradeoff: For a given data size of  $k$  bits:
  - The larger the value of  $n$ , the more errors the algorithm can detect/correct (GOOD)
  - The larger the value of  $n$ , the lower efficiency of the algorithm (BAD)
- Hence there are different approaches to error detection/correction. The best approach depends on many factors like the chance of errors, the user requirements, the protocols, ...

