澳 門 理 工 大 學
Universidade Politécnica de Macau
Macao Polytechnic University

Faculty of Applied Sciences
B.Sc. in Computing

Academic Year 2022/2023 2nd Semester

COMP123 – 121/122
Data Communications

Data Link Control

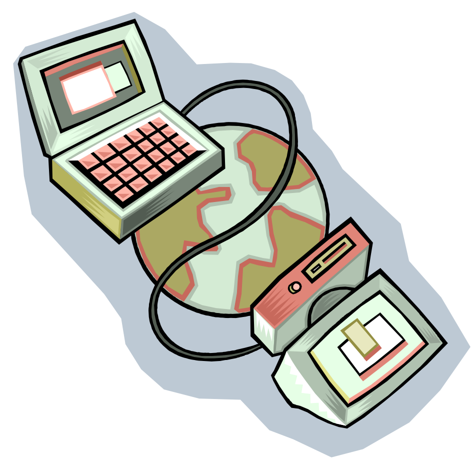
(Flow control, error detection and correction)

The Need for Data Link Control

- Possibility of transmission errors
- Receiver of data may need to regulate the rate at which data arrive
- Synchronization and interfacing techniques are not sufficient
- Need to impose a layer of control in each communicating devices that provides functions (flow and error control) above the physical layer
- So far we have discussed *sending signals over a transmission link*

Data Link Control Protocols

- when sending data, to achieve control, a layer of logic is added above the Physical layer
  - data link control or a data link control protocol
- to manage exchange of data over a link:
  - frame synchronization
  - flow control
  - error control
  - addressing
  - control and data
  - link management

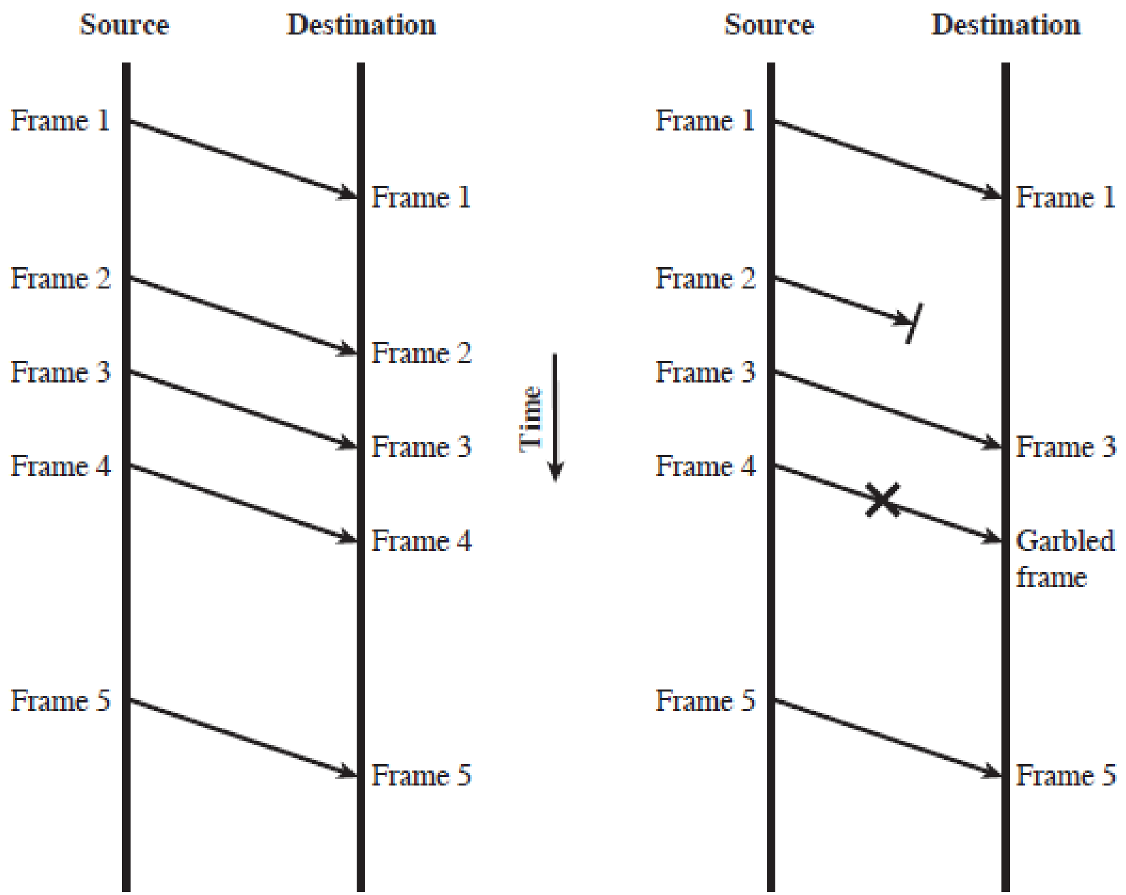


Flow Control

- Ensure sending entity does not overwhelm (crush) receiving entity
  - prevent buffer overflow
- Influenced by:
  - transmission time
    - time taken to emit all bits into medium
  - propagation time
    - time for a bit to traverse the link
- Assumption is all frames are successfully received with no frames lost or arriving with errors

A demo on delay: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/transmission/delay.html
A demo on flow control: http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

Model of Frame Transmission

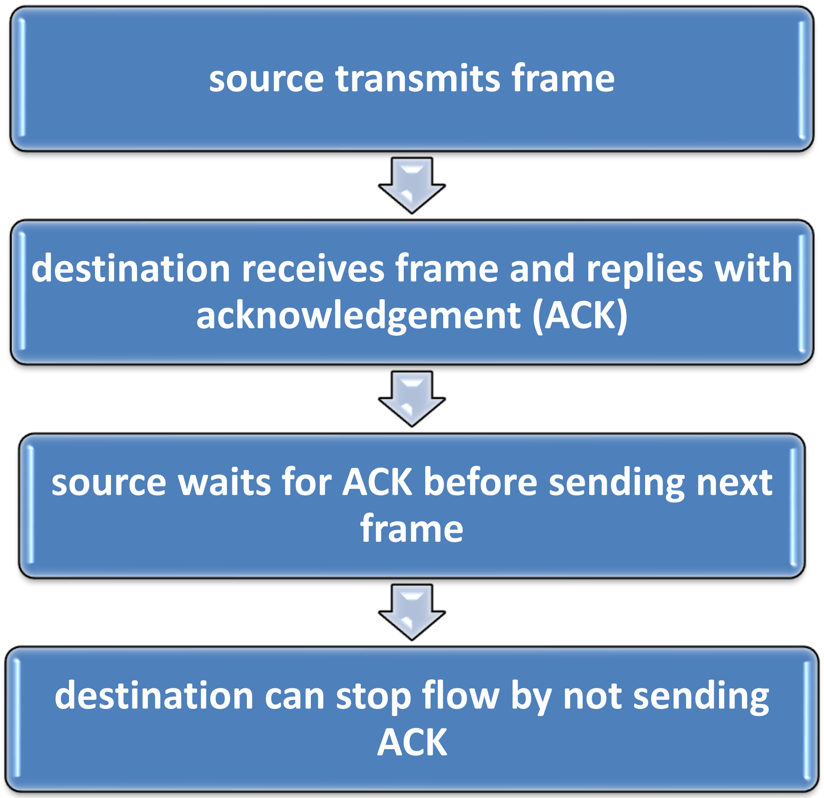


(a) Error-free transmission

(b) Transmission with losses and errors

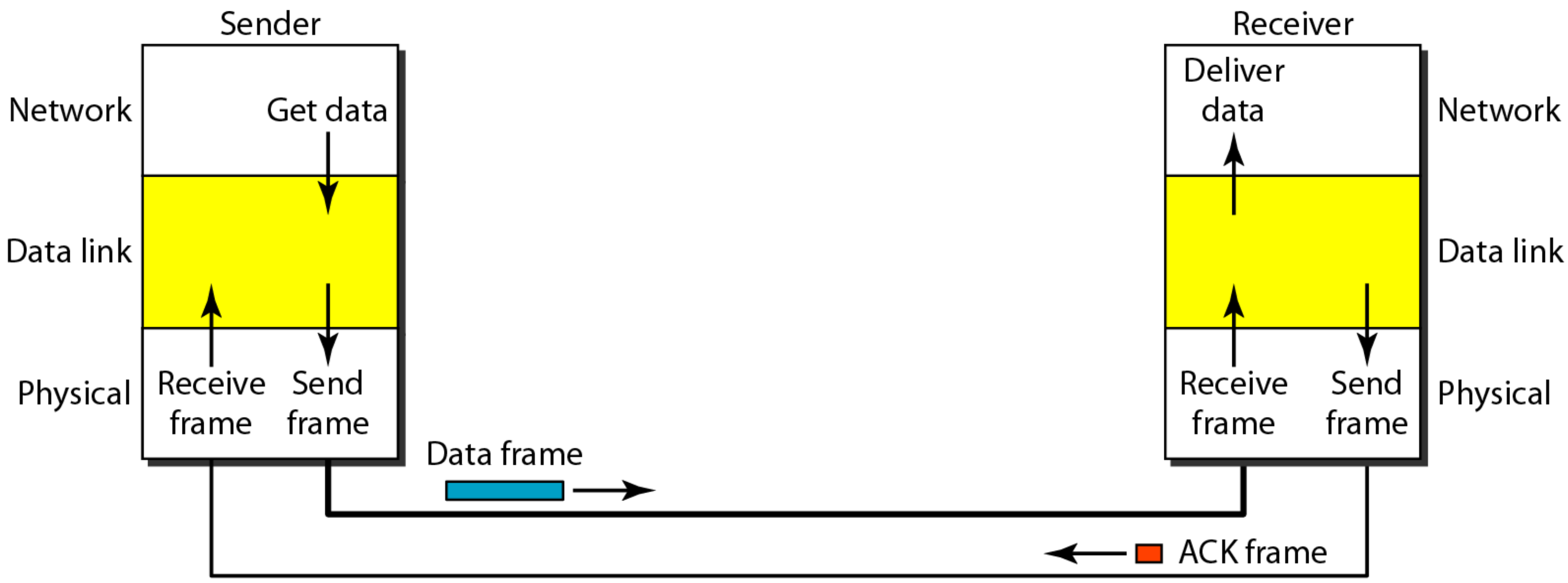
Stop and Wait

- simplest form of flow control

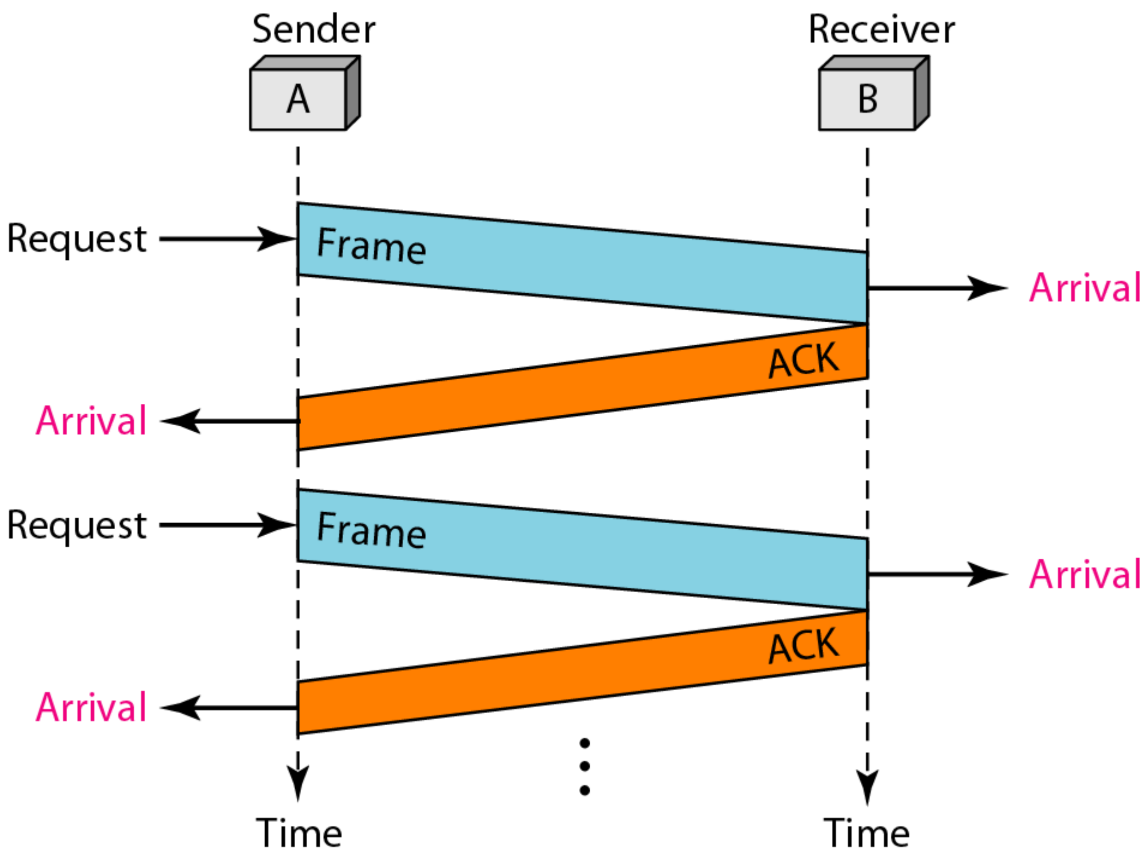


- works well for a message sent in a few large frames
  - stop and wait becomes inadequate if large block of data is split into small frames by source
  - Only one frame at a time can be in transit
  - Sliding Window Flow Control can be used to improved the performance (more on COMP214)

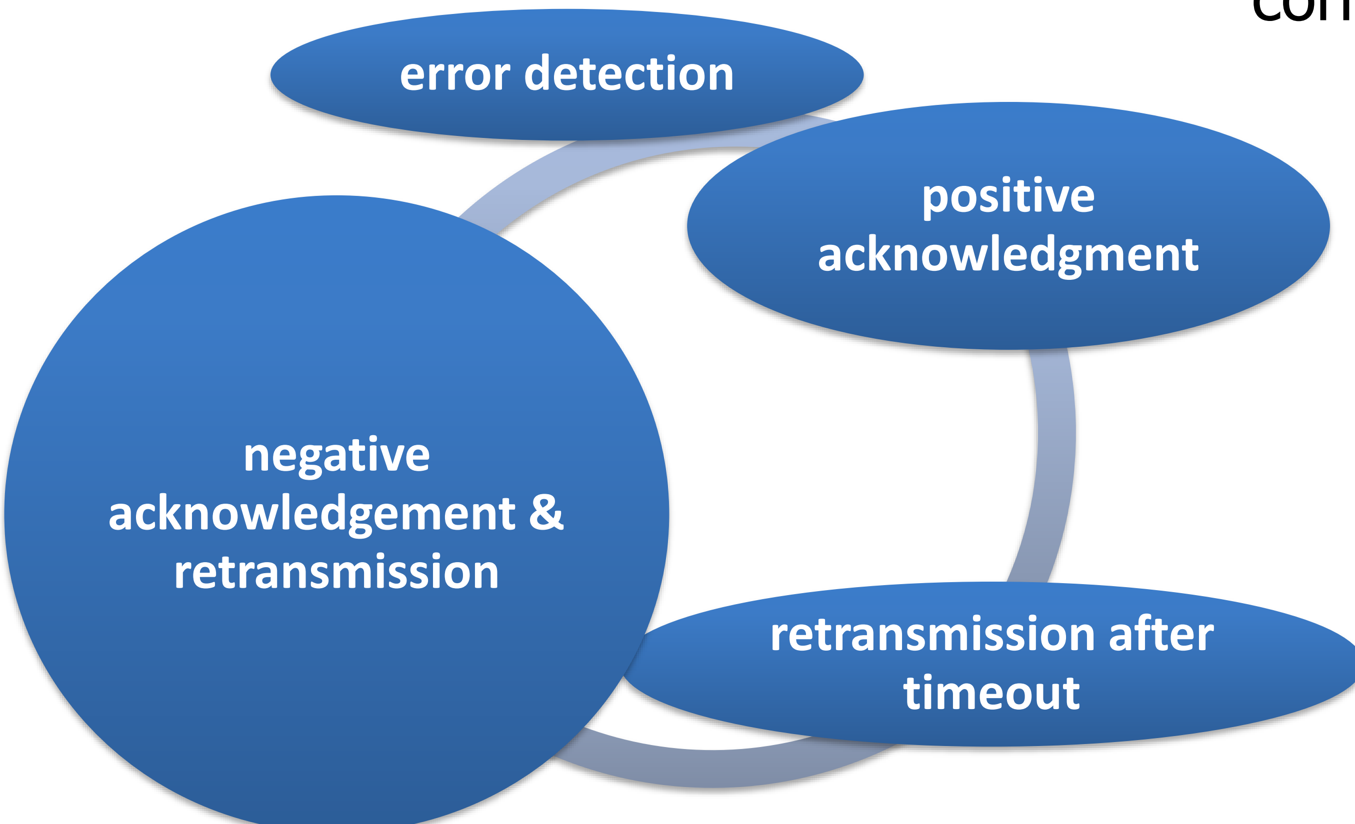
Design of Stop and Wait Protocol



Flow Diagram of Stop and Wait



Error Control Techniques



detection and correction of errors such as:

lost frames

-a frame fails to arrive at the other side

damaged frames

-frame arrives but some of the bits are in error

Error Control Techniques

- **Error detection:** The destination detects frames that are in error and discards those frames.
- **Positive acknowledgment:** The destination returns a positive acknowledgment to successfully received, error-free frames.
- **Retransmission after timeout:** The source retransmits a frame that has not been acknowledged after a predetermined amount of time.
- **Negative acknowledgment and retransmission:** The destination returns a negative acknowledgment to frames in which an error is detected. The source retransmits such frames

Automatic Repeat Request (ARQ)

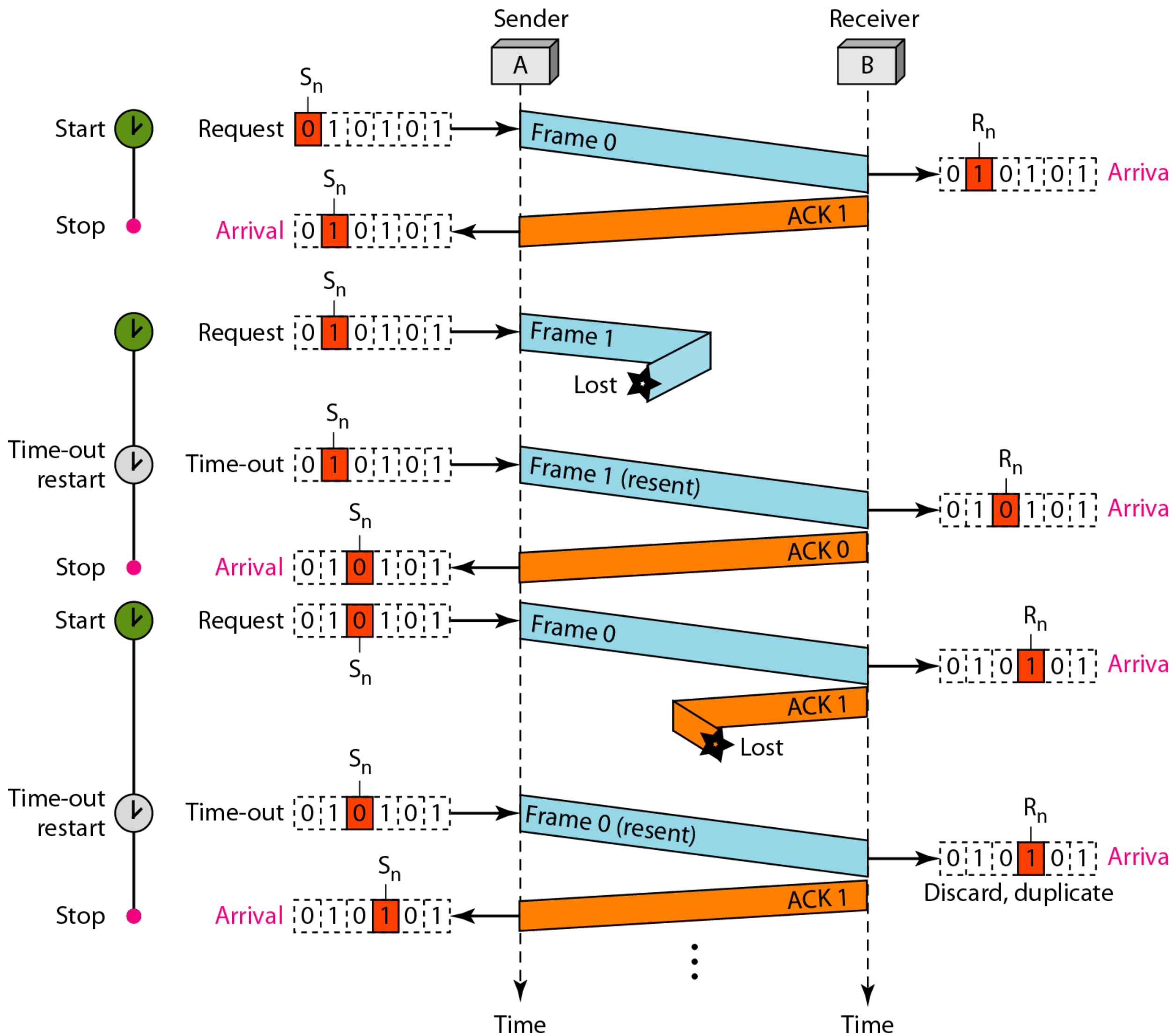
- collective name for error control mechanisms
- effect of ARQ is to turn an *unreliable* data link into a reliable one
- versions of ARQ are:
  - Stop-and-Wait ARQ
  - Sliding Window ARQ (more details in COMP214)

Stop and Wait ARQ

- source transmits single frame
- waits for ACK
    - no other data can be sent until destination’s reply arrives
- if frame received is damaged, discard it
    - transmitter has timeout
    - if no ACK within timeout, retransmit
- if ACK is damaged, transmitter will not recognize
    - transmitter will retransmit
    - receiver gets two copies of frame
    - use alternate numbering and ACK0 / ACK1

Stop and Wait ARQ

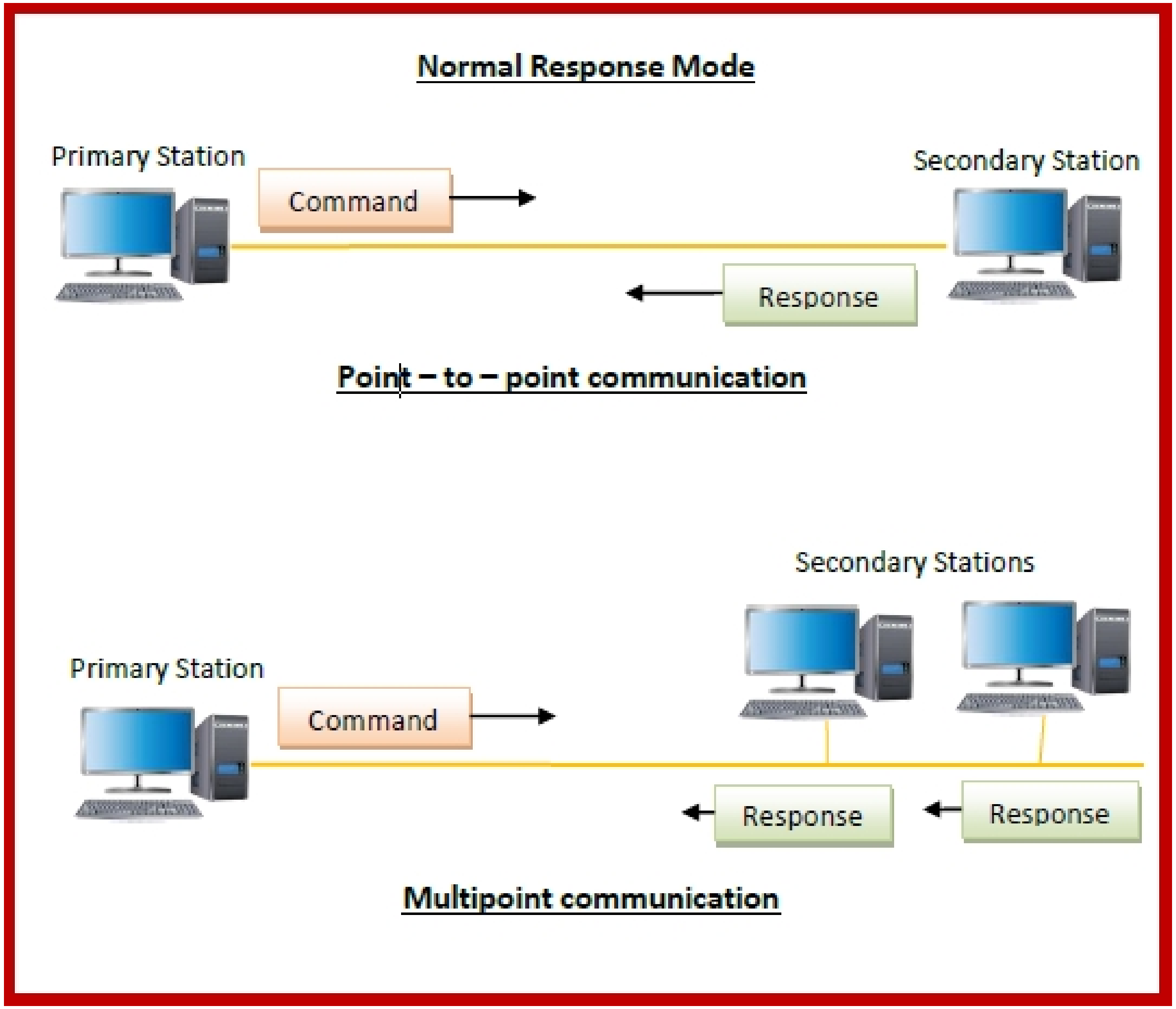
- Pros
  - simplistic
- Cons
  - inefficient



High Level Data Link Control (HDLC)

- an important data link control protocol
- specified as ISO 3309, ISO 4335
- station types:
  - Primary - controls operation of link
  - Secondary - under control of primary station
  - Combined - issues commands and responses
- link configurations
  - Unbalanced - 1 primary, multiple secondary
  - Balanced - 2 combined stations

High Level Data Link Control (HDLC)



Error Detection and Correction

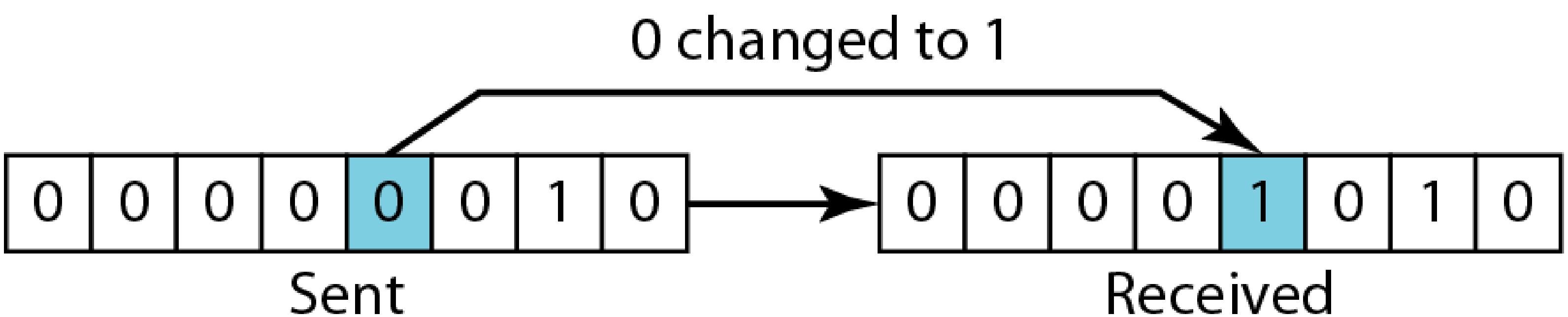
- Recall that error control refers to mechanisms to detect and correct errors that occur in the transmission of frames
- But how do we detect and correct errors?
- Error detection refers to the checking of errors in a frame using coding techniques
- Error correction refers to the correction of errors in a frame using coding techniques

Types of Error

- Data can be corrupted during transmission
- An error occurs when a bit is altered between transmission and reception
  - binary 1 is transmitted and binary 0 is received or binary 0 is transmitted and binary 1 is received
- Two types of errors
  - Single bit errors
  - Burst errors

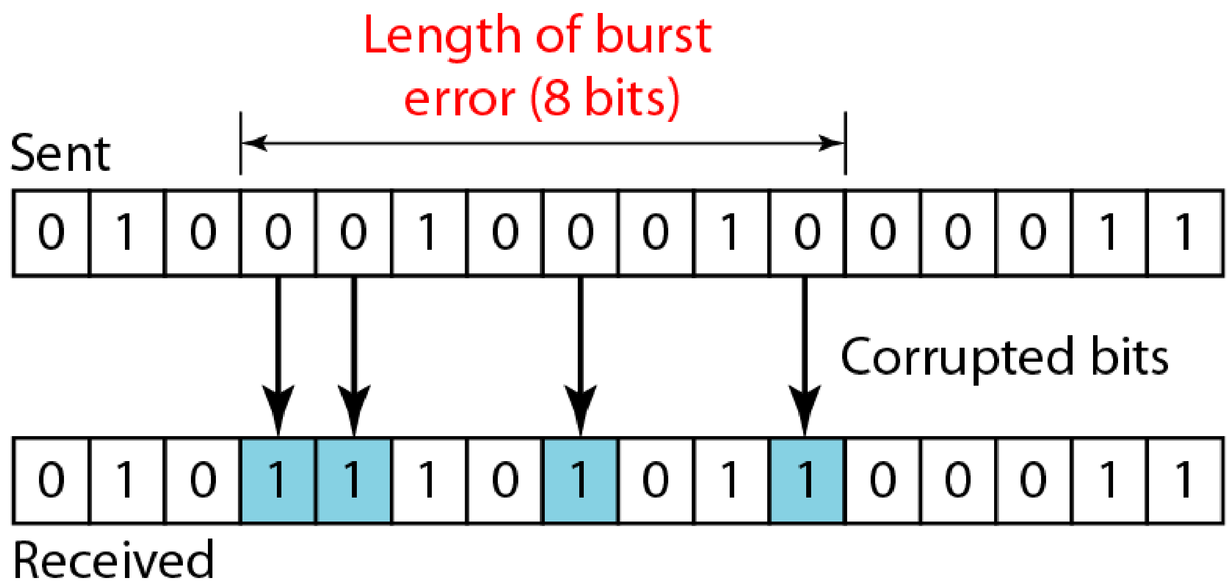
Single Bit Error

- isolated error that alters one bit but not nearby bits
- Caused by white noise



Burst Error

- contiguous sequence of *B* bits where first and last bits and any number of intermediate bits are received in error
- caused by impulse noise or by fading in wireless
- effects greater at higher data rates



Redundancy

- Redundant (extra) bits are used to detect and correct bit errors.
- Redundancy can be built into individual character or into an entire transmitted block.
- Redundant bits are added by the sender and removed by the receiver
- Redundancy reduces the efficiency of transferring data.

Detection versus Correction

- The correction of errors is more difficult than the detection
- In error correction, we need to know the exact number of bit errors and their corresponding locations
- For example, an 8-bit data unit with 1 error $\Rightarrow$ 8 different locations; if 2 errors, $\Rightarrow$ 28 different locations
- What about 2 errors in 100 bits?

$$\boxed{C_2^8 = \frac{8 \times 7}{2} = 28}$$

Review: Modulo-2 Arithmetic

- Adding: 0+0=0 0+1=1 1+0=1 1+1=0
- Subtracting: 0-0=0 0-1=1 1-0=1 1-1=0
- Addition and subtraction give the same results
- XOR (exclusive OR ⊕) operation can be for both addition and subtraction

$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

$0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1$

b. Two bits are different, the result is 1.

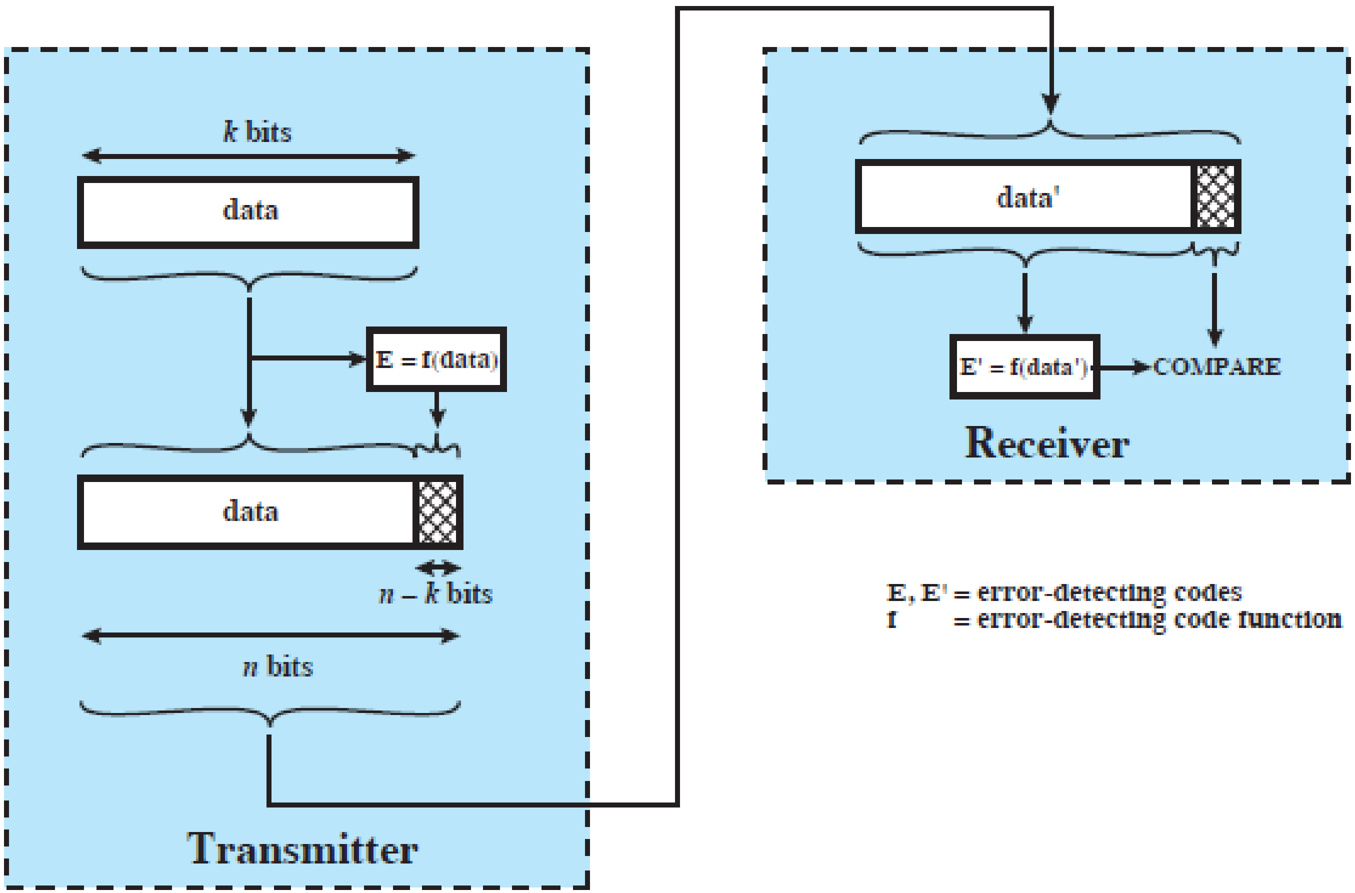
$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & 0 \\ \oplus & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1 & 0 \end{array}$$

c. Result of XORing two patterns

Error Detection

- regardless of design you will have errors
- can detect errors by using an error-detecting code added by the transmitter
    - code is also referred to as *check bits*
- recalculated and checked by receiver
- still chance of undetected error
- Examples: Parity Check, Checksum, Cyclic Redundant Check (CRC)

Error Detection Process



Parity Check

- The simplest error detecting scheme is to append a parity bit to the end of a block of data
- Even parity – even number of 1s
  - typically used for synchronous transmission
- Odd parity – odd number of 1s
  - typically used for asynchronous transmission
- If any <u>even</u> number of bits are inverted due to error, an undetected error occurs

An Example of Even Parity Check Code

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

- Verify that the above parity-check code can detect an odd number of errors.

Efficiency of Parity Checking

- Both sending and receiving devices should use the same parity checking.

- Even and odd parity can detect only an odd number of bit errors. That is , 50% chance of detecting errors.

An Example of Two Dimensional Even Parity Check Code

- Data is organized in a matrix
- 1-parity check bit is added for each row and column
- A parity is added for the last row of parity check bits

2D Even Parity

0	1	1	1	0	1
0	1	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	1
0	0	0	1	1	0

Row parities

Column parities

Parity for the row of column parities

2D Even Parity

$$
\begin{array}{ccccc|c}
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

(b) No errors

$$
\begin{array}{ccccc|c}
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
\hline
0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Row parity error

Column parity error

(c) Correctable single-bit error

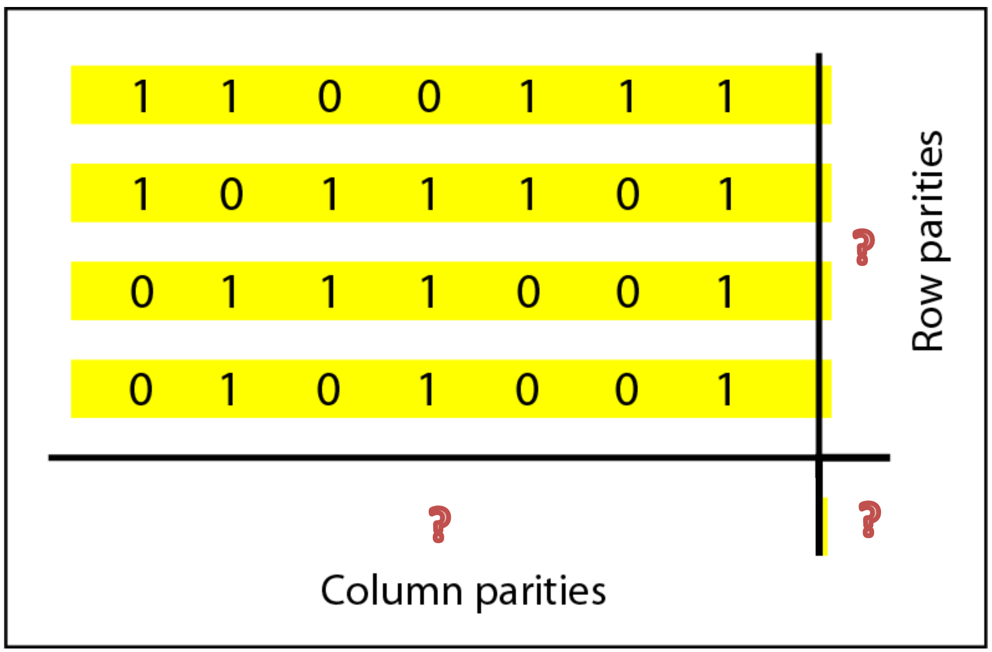
$$
\begin{array}{ccccccc|c}
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
\hline
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

(d) Uncorrectable error pattern

It can detect all 1 and 2 bit errors and correct all 1 bit errors!

An Example of Two Dimensional Even Parity Check Code

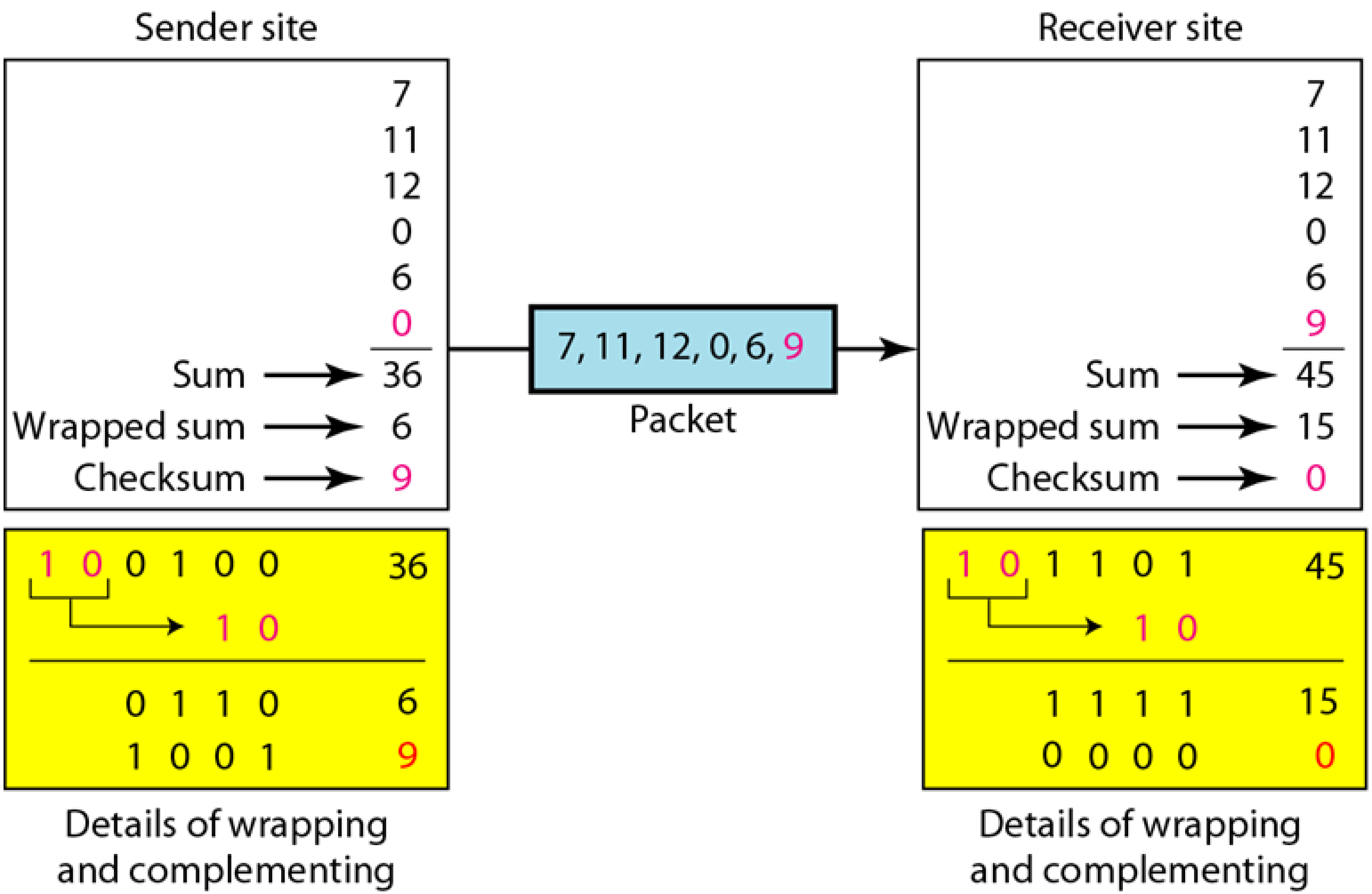
Work out the parities bits.
Can it correct errors?
If so, how many?



Checksums

- Before transmitting a block of data, checksum equals to 0.
- Transmitter takes the binary number representing that character and adds it to the checksum. Checksum is sent with data block.
- Receiver adds up the ones and zeroes that is receives, creating its own checksum.
- The receiver compares two checksums. It they do not match, an error has occurred. The receiver asks the sender to retransmit the data block.
- The effect of errors is to reduce the efficiency, because some blocks must be retransmitted.

An Example of Checksum

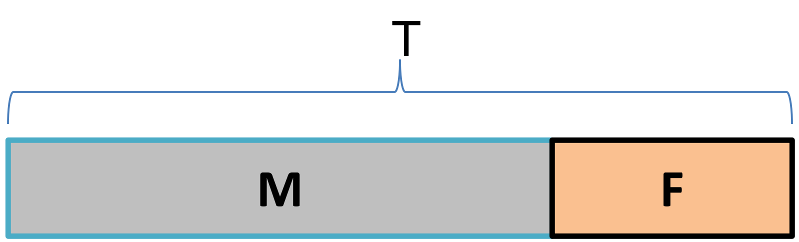


Cyclic Redundancy Check (CRC)

- one of most common and powerful checks
- for block of *k* bits transmitter generates an *n*-*k* bit frame check sequence (FCS)
- Transmits *n* bits which is exactly divisible by some predetermined number
- receiver divides frame by that number
  - if no remainder, assume no error

Cyclic Redundancy Check (CRC)

- **M** *k-bit message*
- **P** *predetermined (n - k + 1)-bit pattern* (the predetermined divisor)
- **F** *(n-k)-bit FCS (Frame Check Sequence),*
  F = Remainder of $(2^{n-k}M)/P$
- **T** *n-bit transmitted frame,*
  $T = 2^{n-k}M + F$



CRC: Example of FCS Generation

Message M = 1010001101 (10 bits)

Pattern P = 110101 (6 bits)

FCS F = to be calculated, should be 5 bits

```
                  1101010110
P-> 110101√101000110100000   ← 2^(n-k)M
           110101
            111011
            110101
              111010
              110101
                111110
                110101
                  101100
                  110101
                   110010
                   110101
                     01110   ← F
```

F is added to $2^{n-k}M$ to give T=101000110101110, which is transmitted

CRC: Example of Error Checking

M = 1010001101 (10 bits)

P = 110101 (6 bits)

F = 01110

```
                   1101010110
P -> 110101√101000110101110   ← T
            110101
             111011
             110101
               111010
               110101
                 111110
                 110101
                   101111
                   110101
                    110101
                    110101
                      00000  ← No Error!
```

No remainder -> no errors

The bit pattern P is chosen depends on the type of errors expected.

Polynomial Codes

- Another way of viewing the CRC process is to express all values as polynomials. So it is also called polynomial codes.

- Ex.

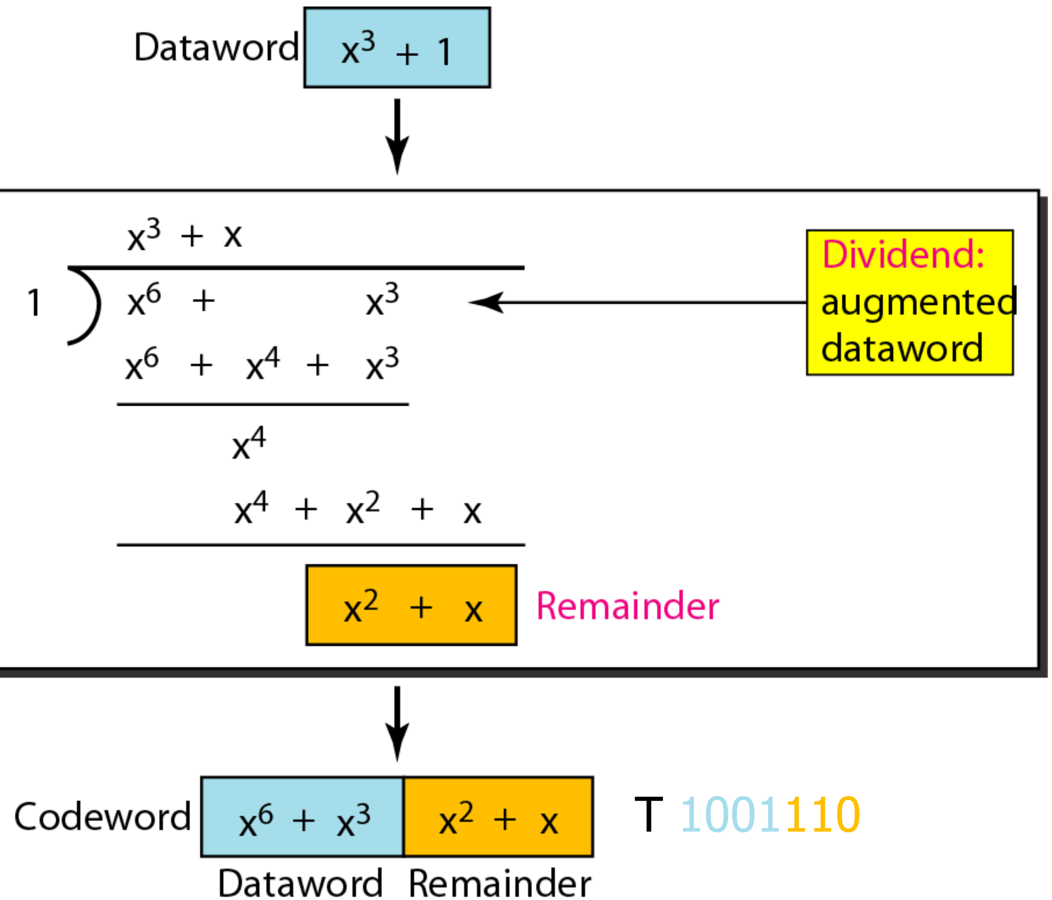
  M = 1010001101 (10 bits)

  P = 110101 (6 bits)

  - $M(X) = X^9 + X^7 + X^3 + X^2 + 1$
  - $P(X) = X^5 + X^4 + X^2 + 1$

An Example of CRC Division using Polynomials

M 1001
P 1011



T 1001110

Polynomial Codes

$$
\begin{array}{rl}
 & X^9 + X^8 + X^6 + X^4 + X^2 + X \quad \leftarrow Q(X) \\
P(X) \rightarrow X^5 + X^4 + X^2 + 1 \,\big) & X^{14} + X^{12} + X^8 + X^7 + X^5 \quad \leftarrow X^5M(X) \\
 & \underline{X^{14} + X^{13} + X^{11} + X^9} \\
 & X^{13} + X^{12} + X^{11} + X^9 + X^8 \\
 & \underline{X^{13} + X^{12} + X^{10} + X^8} \\
 & X^{11} + X^{10} + X^9 + X^7 \\
 & \underline{X^{11} + X^{10} + X^8 + X^6} \\
 & X^9 + X^8 + X^7 + X^6 + X^5 \\
 & \underline{X^9 + X^8 + X^6 + X^4} \\
 & X^7 + X^5 + X^4 \\
 & \underline{X^7 + X^6 + X^4 + X^2} \\
 & X^6 + X^5 + X^2 \\
 & \underline{X^6 + X^5 + X^3 + X} \\
 & X^3 + X^2 + X \leftarrow R(X)
\end{array}
$$

Polynomial Codes

- The choice of *P* depends on the type of error expected.
- It can be shown that all of the following errors are detectable.
  - All single-bit errors
  - All double-bit errors, as long as P(X) has a factor with at least three terms
  - Any odd number of errors, as long as P(X) contains a factor (X + 1)
  - Any burst error for which the length of the burst is less than the length of the CRC
  - Most large burst errors

Four Widely Used P(X)

1. $CRC\text{-}12 = X^{12} + X^{11} + X^3 + X^2 + X + 1$
   - Used for transmitting 6-bit characters

2. $CRC\text{-}16 = X^{16} + X^{15} + X^2 + 1$
   - Used for transmitting 8-bit characters in U.S.A.

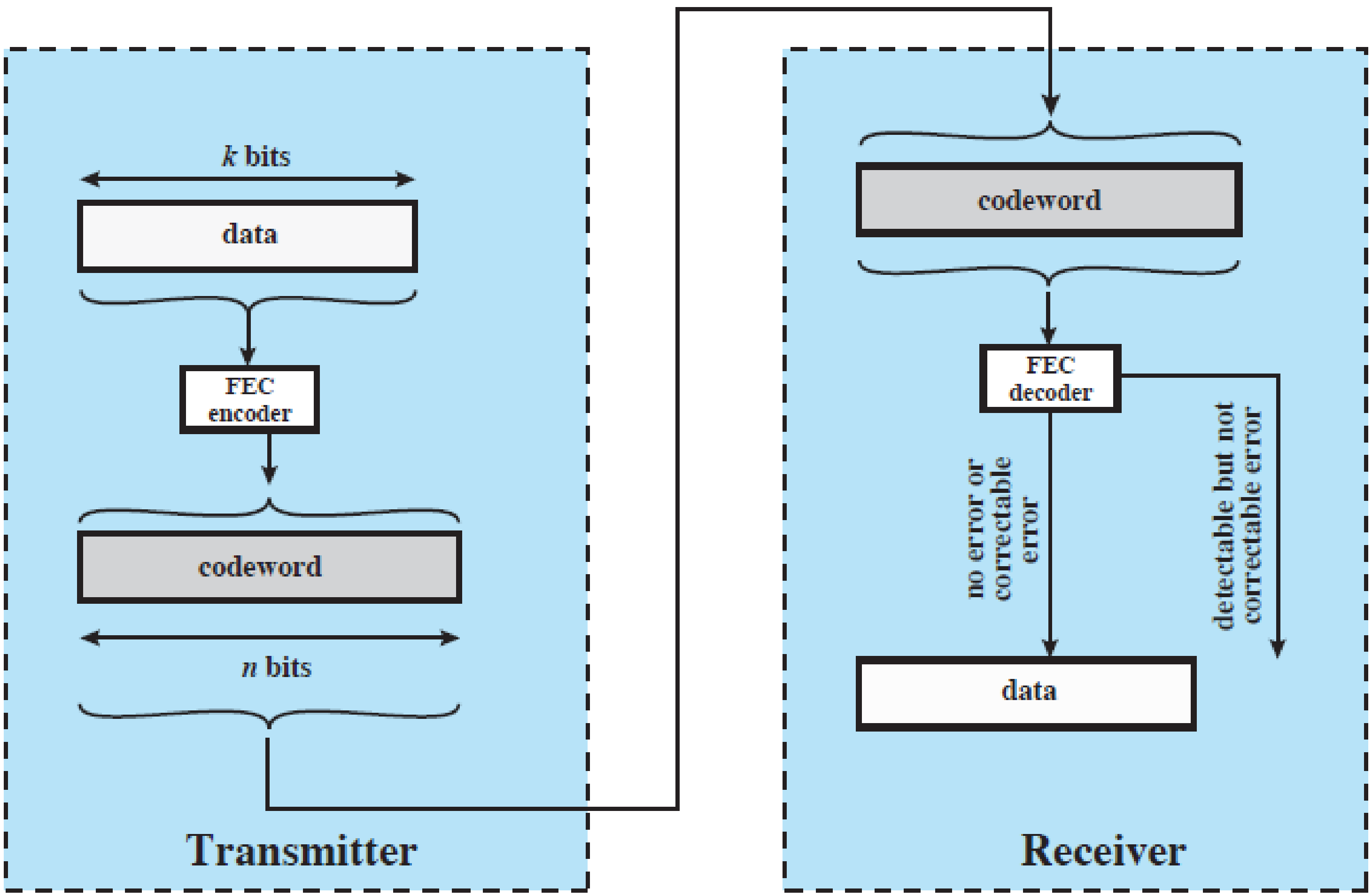
3. $CRC\text{-}CCITT = X^{16} + X^{12} + X^5 + 1$
   - Used for transmitting 8-bit characters in Europe
   - It has been chosen as the international standard after much theoretical work and simulation.

4. $CRC\text{-}32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
   - The IEEE-802 committee uses this as Local Area Network standard.

Error Correction

- correction of detected errors usually requires data block to be *retransmitted*
- not appropriate for wireless applications
  - bit error rate is high causing lots of retransmissions
  - propagation delay long (satellite) compared with frame transmission time, resulting in retransmission of frame in error plus many subsequent frames
- need to correct errors on basis of bits received
- codeword
  - on the transmission end each k-bit block of data is mapped into an n-bit block (n > k) using a forward error correction (FEC) encoder

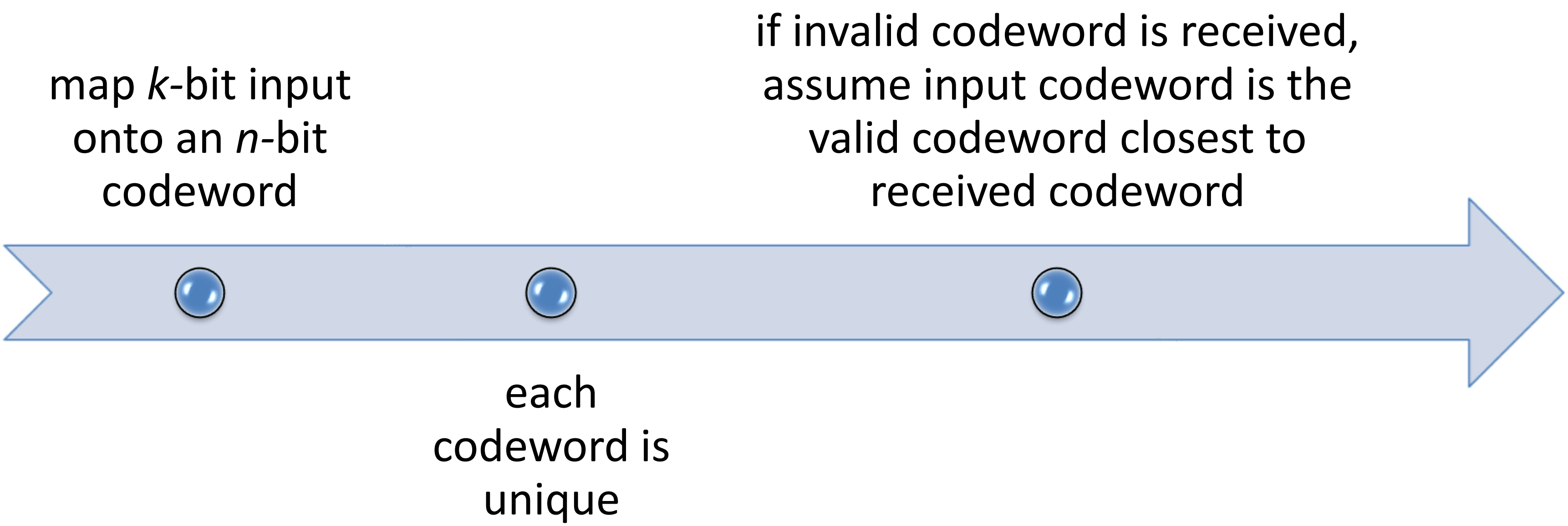
Error Correction Process



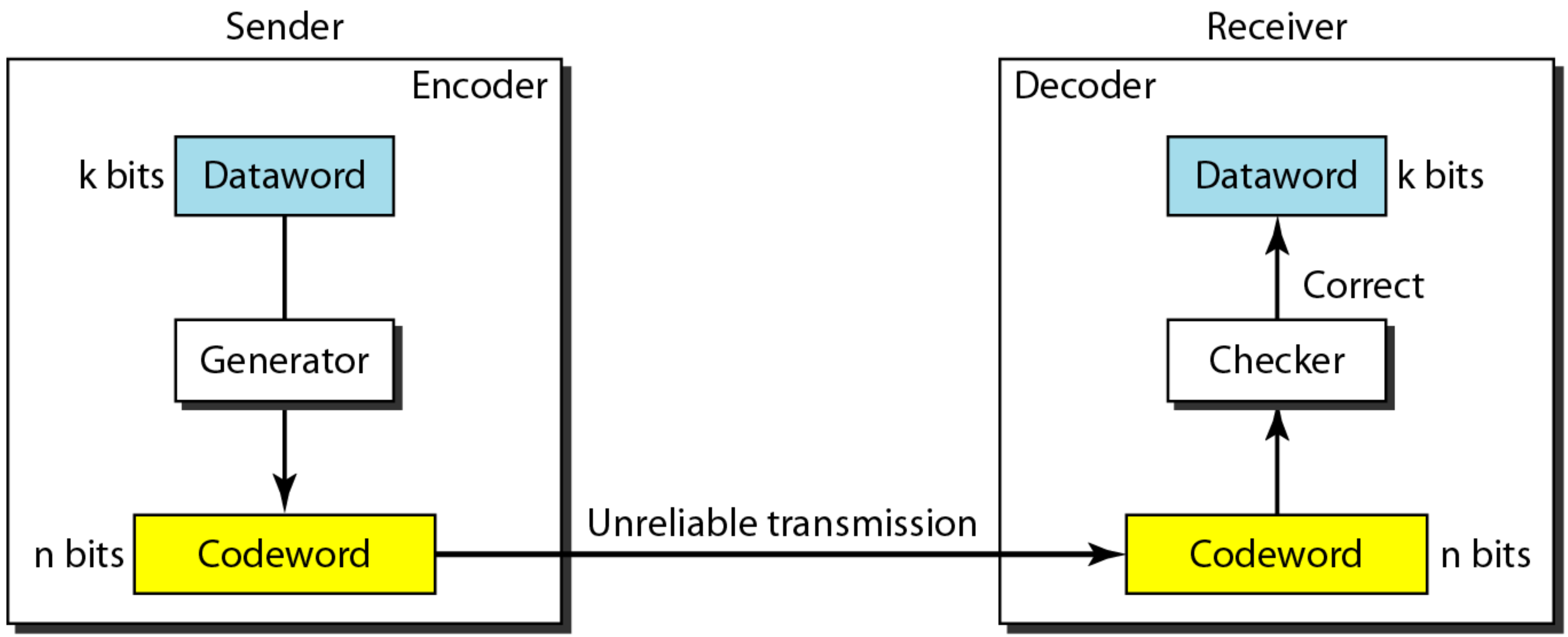
How Error Correction Works

- adds redundancy to transmitted message
  - redundancy makes it possible to deduce original message despite some errors

- block error correction code



Structure of encoder and decoder in error correction



Forward Error Correction (FEC)

- Using extra bits to detect and correct errors
- The ratio between the length of data (k) and the length of codeword (n) is called the code rate $r=k/n$

- Hamming Code is an example. It is devised by Richard Hamming.

- Other examples: covolutional code, turbo code and low density parity check (LDPC) code, etc..

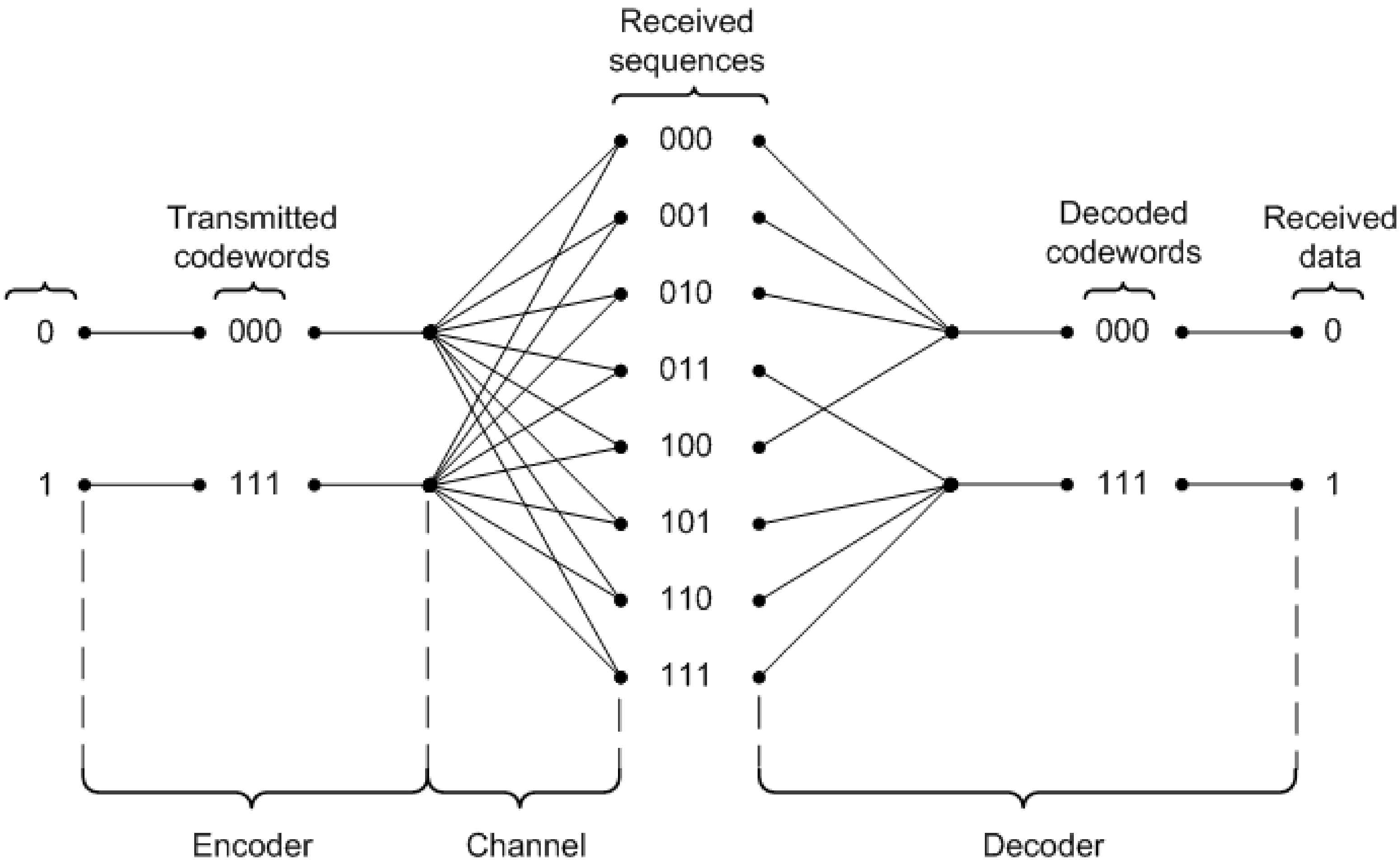
FEC versus Retransmission

- Two methods of error correction: FEC and Retransmission
- FEC is the process in which the receiver tries to guess the message by using redundant bits
- Correction by retransmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message

Hamming Distance

- The Hamming distance between two bit patterns is the number of bits that are different in the pattern.
- For example, A = 1000001, B = 1000010, C = 1000011, the Hamming distance between A and C is 1, the Hamming distance between A and B is 2.
- If there is a bit error, A will more likely to change to C, then change to B.
- Hamming suggests that all bit patterns used in transmission should have the maximum Hamming distance.
- For example, if all bit patterns (codewords) have 2 Hamming distance, then any 1 bit error will be immediately detected.

Repetition Code



Hamming Code

- Allow the receiving device not only to detect a single bit error, but also to correct it with 100% accuracy.

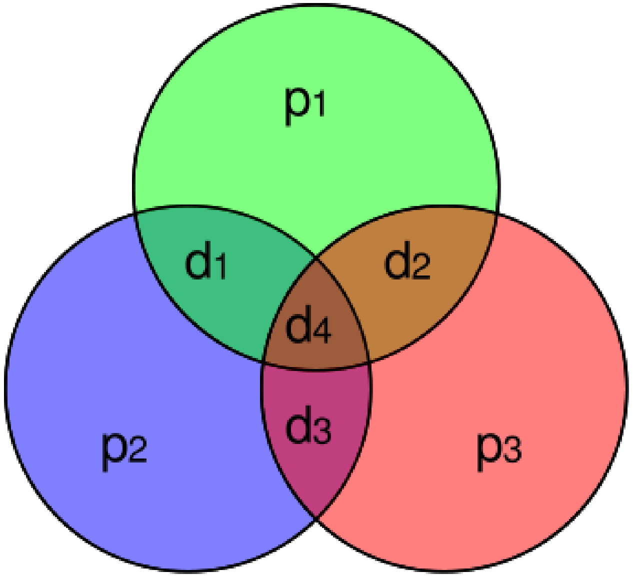
- Need a lot of check (parity) bits. Check bits are placed among data bits. Each check bit covers a number of data bits.

Hamming Code

- Check/Parity bits are placed in the power of 2's positions.

	D4	D3	D2	P3	D1	P2	P1
bit position	***7***	***6***	***5***	***4***	***3***	***2***	***1***

- $P1 = D1 \oplus D2 \oplus D4$
- $P2 = D1 \oplus D3 \oplus D4$
- $P3 = D2 \oplus D3 \oplus D4$



- ***Note that***
  - ***D1*** is covered by ***P1, P2***
  - ***D2*** is covered by ***P1, P3***
  - ***D3*** is covered by ***P2, P3***
  - ***D4*** is covered by ***P1, P2, P3***

- What is the code rate of this Hamming code?
- What is the minimum Hamming distance of this code?

An Example of (7,4) Hamming Code

- Code the data 0110, using the Hamming coding method for transmission.

$D1 = 0, \; D2 = 1, \; D3 = 1, \; D4 = 0$

$$P1 = D1 \oplus D2 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$
$$P2 = D1 \oplus D3 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$
$$P3 = D2 \oplus D3 \oplus D4 = 1 \oplus 1 \oplus 0 = 0$$

Answer: The hamming code is 0 1 1 0 0 1 1

Hamming Code Decoding (1/2)

- If receive the pattern 0 1 0 0 0 1 1, check it for errors and produce the correct data.

Received: *P1 = 1, P2 = 1, P3 = 0*

Calculated:

$$P1 = D1 \oplus D2 \oplus D4 = 0 \oplus 0 \oplus 0 = 0$$
$$P2 = D1 \oplus D3 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$
$$P3 = D2 \oplus D3 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$

- P1 and P3 are not correct , comparing with the received parity bits.
- P1 and P3 cover together are D2 and D4.
- If D4 is not correct, P2 should not correct, but P2 is correct, so D4 is correct.
- So D2 is the error bit.

- Answer: The data should be 0 1 1 0

  The pattern should be 0 1 1 0 0 1 1

Hamming Code Decoding (2/2)

- If receive the pattern 0 1 1 1 0 1 1, check it for errors and produce the correct data.

Received: $P1 = 1, P2 = 1, P3 = 1$

Calculate:

$$P1 = D1 \oplus D2 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$
$$P2 = D1 \oplus D3 \oplus D4 = 0 \oplus 1 \oplus 0 = 1$$
$$P3 = D2 \oplus D3 \oplus D4 = 1 \oplus 1 \oplus 0 = 0$$

- P3 is not correct.
- P3 covers D2, D3, and D4.
- If D2 is not correct, P1 should be not correct.
- If D3 is not correct, P2 should be not correct.
- If D4 is not correct, P1, P2 should be not correct.
- But P1 and P2 are correct, so D2, D3, D4 are all correct.
- P3 is the error bit

- Answer: The correct data is 0110

  The pattern should be 0 1 1 0 0 1 1

Error Bit Position for Hamming Code

Incorrect Check Bits	Error Bit Position
P1	*1 (P1)*
P2	*2 (P2)*
P1 and P2	*3 (D1)*
P3	*4 (P3)*
P1 and P3	*5 (D2)*
P2 and P3	*6 (D3)*
P1, P2, and P3	*7 (D4)*

(15,11) Hamming Code

- We have just discussed Hamming code with 4 data bits and 3 check bits.
- Another Hamming code with 11 data bits, 4 check bits can also be generated.

D11	D10	D9	D8	D7	D6	D5	P4	D4	D3	D2	P3	D1	P2	P1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

- $P1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \oplus D9 \oplus D11$
- $P2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \oplus D10 \oplus D11$
- $P3 = D2 \oplus D3 \oplus D4 \oplus D8 \oplus D9 \oplus D10 \oplus D11$
- $P4 = D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 \oplus D10 \oplus D11$

Summary

- data link protocols
  - flow control
    - stop-and-wait, sliding window, ACK frame
  - error control
    - Lost frame, damaged frame
    - Stop-and-wait, sliding windows ARQs
  - HDLC
- Error detection and correction
  - Error detection
    - Parity, check sum, CRC
  - Error correction
    - Repetition, Hamming

Implementation of Polynomial Codes

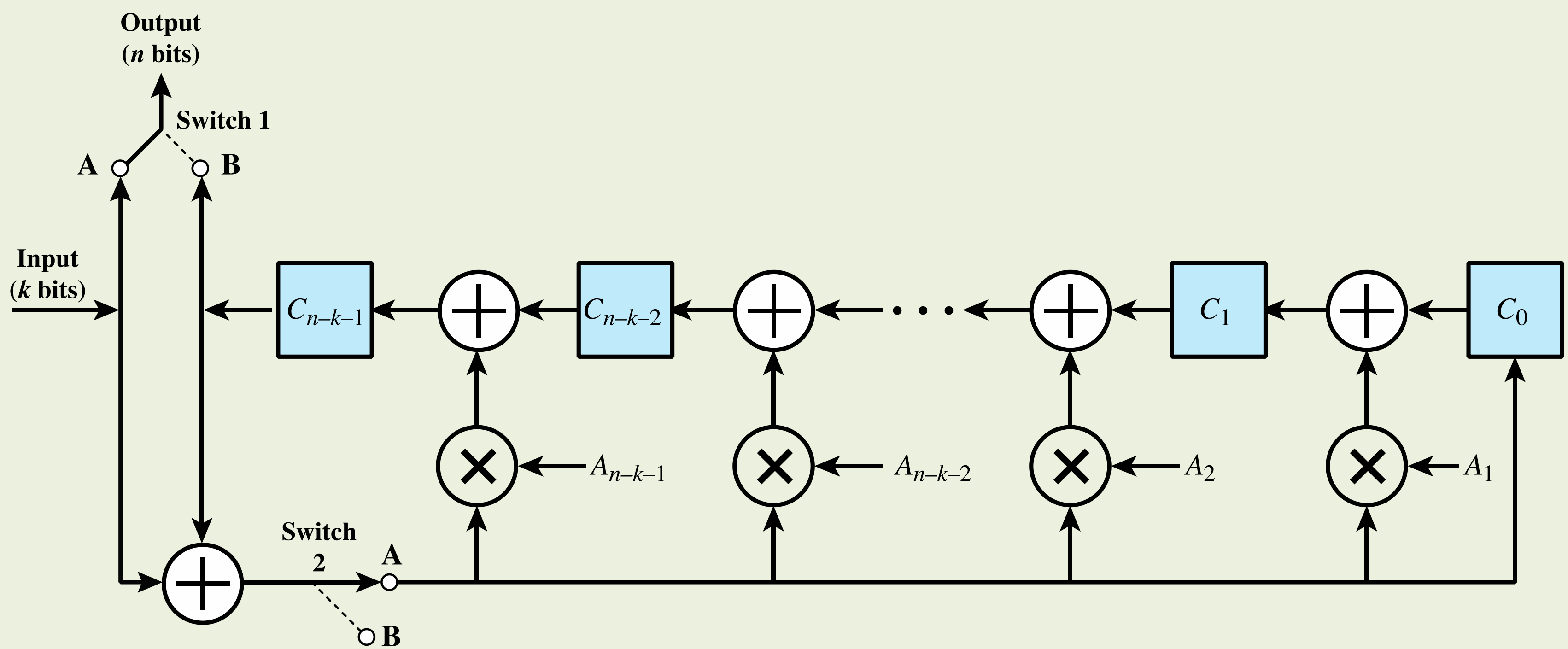
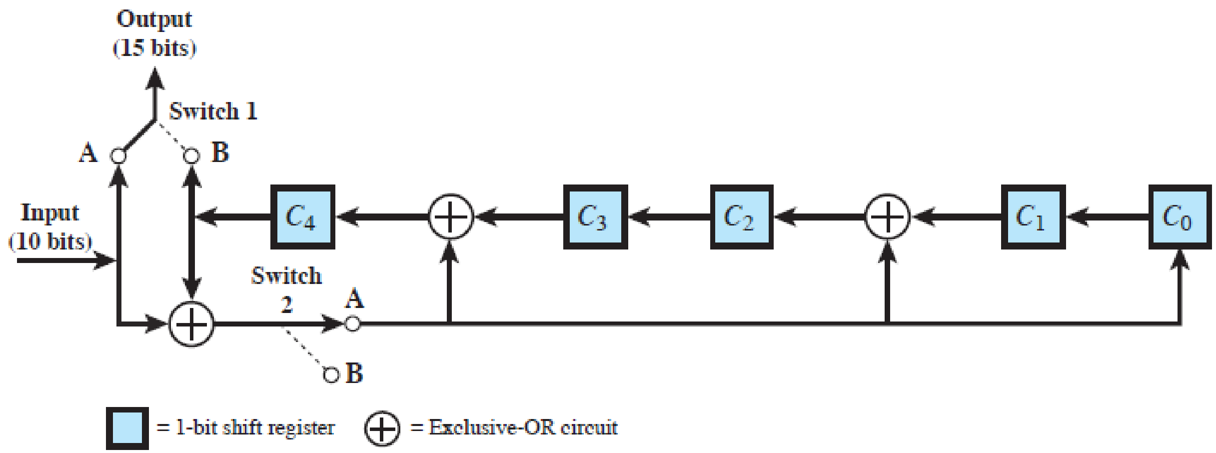


Figure 6.7 General CRC Architecture to Implement Divisor
$(1 + A_1X + A_2X^2 + \ldots + A_{n-k-1}X^{n-k-1} + X^{n-k})$

Implementation of Polynomial Codes



(a) Shift-register implementation

	C_4	C_3	C_2	C_1	C_0	$C_4 \approx C_3 \approx I$	$C_4 \approx C_1 \approx I$	$C_4 \approx I$	I = input
Initial	0	0	0	0	0	1	1	1	1
Step 1	1	0	1	0	1	1	1	1	0
Step 2	1	1	1	1	1	1	1	0	1
Step 3	1	1	1	1	0	0	0	1	0
Step 4	0	1	0	0	1	1	0	0	0
Step 5	1	0	0	1	0	1	0	1	0
Step 6	1	0	0	0	1	0	0	0	1
Step 7	0	0	0	1	0	1	0	1	1
Step 8	1	0	0	0	1	1	1	1	0
Step 9	1	0	1	1	1	0	1	0	1
Step 10	0	1	1	1	0				

Message to be sent

(b) Example with input of 1010001101

Figure 6.5 Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$