COMP 3721
Introduction to Data Communications

07a - Week 7 - Part 1

COMP 3721 数据通信导论

第7周 - 第1部分 07a

Learning Outcomes

- By the end of this lecture, you will be able to
  - Explain what are the types of errors.
  - Distinguish between error detection and error correction.
  - Explain how block coding and hamming distance work.
  - Explain how cyclic codes work.
  - Explain how to calculate a checksum.
  - Explain FEC (Forwarding Error Correction) techniques.

学习目标

- 在本讲座结束时，您将能够
  - 解释错误的类型有哪些。
  - 区分错误检测与错误纠正。
  - 解释分组编码和汉明距离的工作原理。
  - 解释循环码的工作原理。
  - 解释如何计算校验和。
  - 解释前向纠错（FEC）技术。

Introduction

- Networks must be able to transfer data from one device to another with **acceptable accuracy**.
  - The data received **must be identical** to the data transmitted.
- Data may become **corrupted** in passage (one or more bits may be modified).
- A small level of error may be tolerated by some applications.

简介

- 网络必须能够在设备之间传输数据，并具有**可接受的准确性**。
  - 接收到的数据**必须与**发送的数据完全相同。
- 数据在传输过程中可能**出错**（一个或多个比特可能被修改）。
- 某些应用程序可容忍一定程度的错误。

Introduction

- Networks must be able to transfer data from one device to another with **acceptable accuracy**.
  - The data received **must be identical** to the data transmitted.
- Data may become **corrupted** in passage (one or more bits may be modified).
- A small level of error may be tolerated by some applications.
- The approach of most link-layer protocols for error control → simply **discard the frame** and let the **upper-layer protocols** handle the **retransmission** of the frame.

简介

- 网络必须能够以**可接受的准确性**将数据从一个设备传输到另一个设备。
  - 接收到的数据**必须与**发送的数据完全相同。
- 数据在传输过程中可能**受损**（一个或多个比特可能被修改）。
- 某些应用可以容忍一定程度的错误。
- 大多数链路层协议用于错误控制的方法→ 只是**丢弃该帧**，并让**上层协议**负责处理该帧的**重传**。

Types of Errors

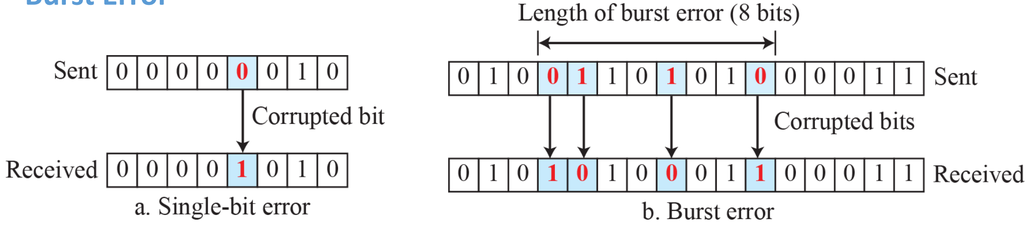
- **Interference** can change the shape of the signal.
  - Bits are subject to unpredictable changes.

错误类型

- **干扰** 可以改变信号的形状。
  - 比特可能发生不可预测的变化。

Types of Errors

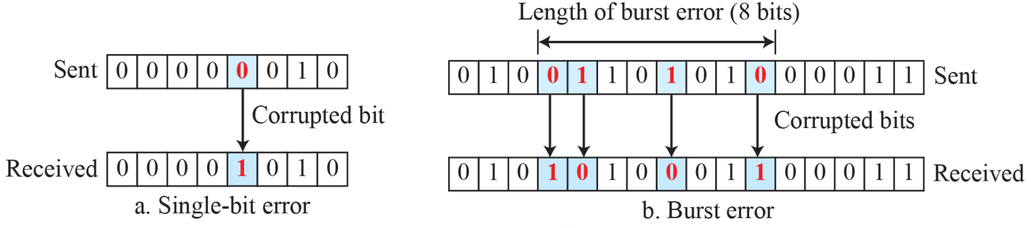
- **Interference** can change the shape of the signal.
  - Bits are subject to unpredictable changes.
- **Single-bit error** and **Burst error**
  - Only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1 → **single-bit error**
  - 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1 → **Burst Error**



More likely to happen. Why?

错误类型

- **干扰**可能会改变信号的形状。
  - 比特可能会发生无法预测的变化。
- **单比特错误**和**突发错误**
  - 给定数据单元（如字节、字符或数据包）中只有一个比特从1变为0或从0变为1 → **单比特错误**
  - 2 数据单元中有两个或更多比特从1变为0或从0变为1 → **突发错误**



更有可能发生。为什么?

Number of Bits affected by the Noise

Number of bits affected by the noise depends on the **data rate** and the **duration of noise**.

受噪声影响的比特数

噪声影响的比特数取决于**数据速率**和**噪声持续时间**。

Example

- Assume we are sending data and there is a noise of 0.001 second.
- How many bits are affected by this noise if
  a) The data rate is 1 kbps.
  b) The data rate is 1 Mbps.

示例

- 假设我们正在传输数据，且存在 0.001 秒的噪声。
- 如果发生这种噪声，会有多少位受到影响?
  a) 数据速率为 1 kbps。
  b) 数据速率为 1 Mbps。

Example

- Assume we are sending data and there is a noise of 0.001 second.
- How many bits are affected by this noise if
  a) The data rate is 1 kbps.
  b) The data rate is 1 Mbps.

- **Answer:**
  a) 1000 × 0.001 = **1 bit**
  b) 1000000 × 0.001 = **1000 bits**

示例

- 假设我们正在传输数据，且存在 0.001 秒的噪声。
- 如果数据速率为以下情况，该噪声会影响多少位?
  a) 数据速率为 1 kbps。
  b) 数据速率为 1 Mbps。

- **答案：**
  a) 1000 × 0。001 = **1 bit**
  b) 1000000 × 0。001 = **1000 位**

Error Detection vs. Error Correction

- **Error detection**
  - Only looking to see if any error has occurred.
  - The number of corrupted bits (type of error) is not important.
- **Error correction**
  - The **exact number of corrupted bits** and **their location** in the message need to be identified.
  - The number of errors and the size of the message are important factors.

- The **correction** of errors is **more difficult** than the **detection**.

错误检测与错误纠正

- **错误检测**
  - 仅查看是否发生了任何错误。
  - 出错比特的数量（错误类型）并不重要。
- **错误纠正**
  - 必须确定消息中 **损坏比特的确切数量** 以及 **它们的位置**。
  - 错误的数量和消息的大小是重要因素。

- 纠正**错误**比检测错误**更加困难**。

Redundancy for Error Control

- Sending some **extra bits** with our data.
- These redundant bits are **added by the sender** and **removed by the receiver**.
- Redundant bits allow the **receiver** to **detect** or **correct** corrupted bits.
- Redundancy is achieved through **coding schemes**.
- The sender adds redundant bits which **have a relationship** with the actual data.

用于错误控制的冗余

- 发送一些 **额外的比特** 与我们的数据一起。
- 这些冗余比特由 **发送方添加** ，并由 **接收方移除**。

- 冗余比特使 **接收方** 能够 **检测** 或 **纠正**被损坏的比特。
- 通过 **编码方案** 实现冗余。
- 发送方添加具有与实际数据 **某种关系的冗余比特**。

Block Coding

- **Dataword**
  - The message is divided into blocks, each of *k* bits.

块编码

- **数据字**
  - 消息被划分为若干块，每块为*k*位。

Block Coding

- **Dataword**
  - The message is divided into blocks, each of k bits.
- **Codeword**
  - r redundant bits are added to each block to make the length $n = k + r$

块编码

- **数据字**
  - 消息被划分为多个块，每个块为k位。
- **码字**
  - r个冗余位被添加到每个块中，使长度变为$n = k + r$

Block Coding

- **Dataword**
  - The message is divided into blocks, each of k bits.
- **Codeword**
  - r redundant bits are added to each block to make the length $n = k + r$
- 2^k datawords < 2^n codewords

块编码

- **数据字**
  - 消息被划分为若干块，每块为k位。
- **码字**
  - r个冗余位被添加到每个块中，使长度变为$n = k + r$
- 2_k 数据字 < 2_n 码字

Block Coding

- **Dataword**
  - The message is divided into blocks, each of k bits.
- **Codeword**
  - r redundant bits are added to each block to make the length $n = k + r$
- 2^k datawords < 2^n codewords
- The **block coding** process is **one-to-one**.
  - The **same dataword** is always encoded as the **same codeword**.
  - $(2^n - 2^k)$ codewords are not used (invalid codes) → the **trick** in error detection

块编码

- **数据字**
  - 消息被划分为若干块，每块为k位。
- **码字**
  - 向每个块添加r个冗余位，使长度变为$n = k + r$
- 2_k个数据字 < 2_n个码字
- 该**分组编码**过程是**一对一**的。
  - 相同的**数据字**始终被编码为相同的**码字**。
  - $(2_n - 2_k)$ 个码字未被使用（无效代码）→ 错误检测中的**技巧**在于

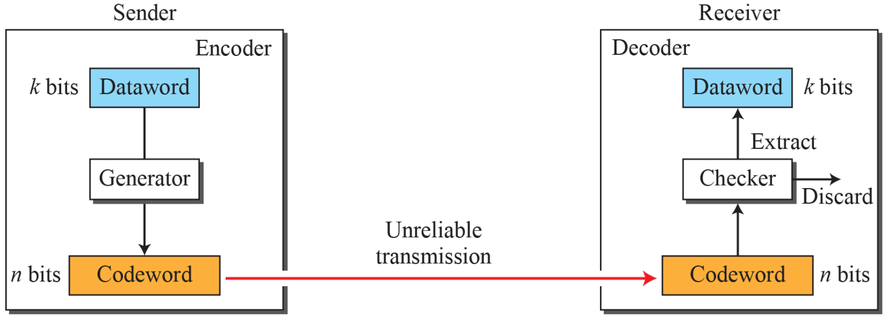
Block Coding

- **Dataword**
  - The message is divided into blocks, each of k bits.
- **Codeword**
  - r redundant bits are added to each block to make the length $n = k + r$
- 2^k datawords < 2^n codewords
- The **block coding** process is **one-to-one**.
  - The **same dataword** is always encoded as the **same codeword**.
  - $(2^n - 2^k)$ codewords are not used (invalid codes) → the **trick** in error detection
- **Error detection** by **receiver** (3 conditions)
  1. The receiver has (or can find) a list of valid codewords.
  2. The original codeword has changed to an invalid one and it'll be discarded.
  3. The original codeword changes to a valid one! → makes the error **undetectable**

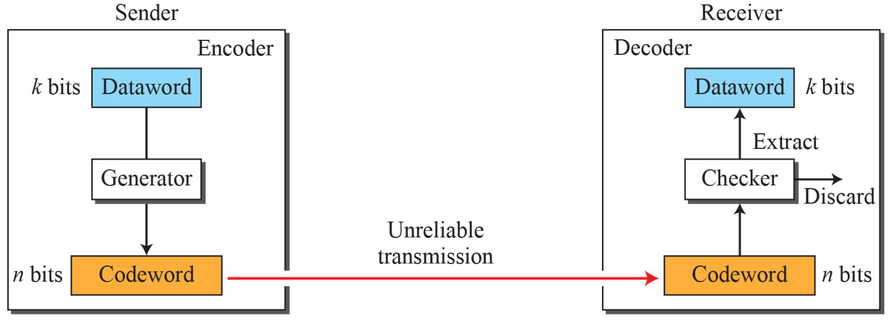
块编码

- **数据字**
  - 消息被划分为多个块，每个块为k位。
- **码字**
  - 向每个块中添加r个冗余位，使长度变为$n = k + r$
- 2_k个数据字 < 2_n个码字
- 该**分组编码**过程是**一对一**。
  - 相同的**数据字**始终被编码为相同的**码字**。
  - $(2_n - 2_k)$ 个码字未被使用（无效码）→ 错误检测中的**技巧**在于
- **错误检测**由**接收方**执行（三种情况）
  1. 接收方拥有（或能够找到）一份有效码字的列表。
  2. 原始码字已变为无效码字，因此将被丢弃。
  3. 原始密码更改为一个有效的密码！→ 使得错误**无法被检测到**

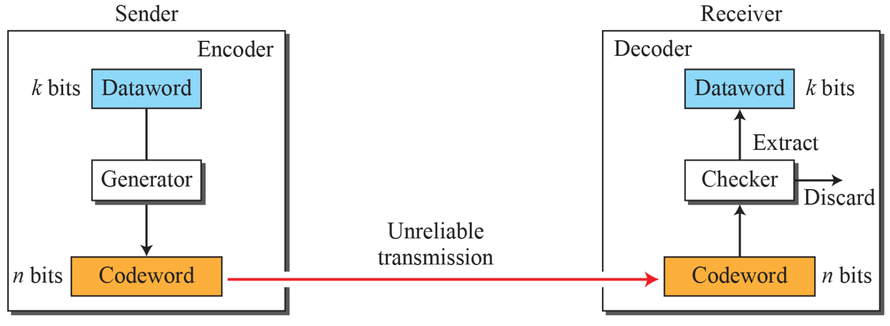
Error Detection in Block Coding



分组编码中的错误检测

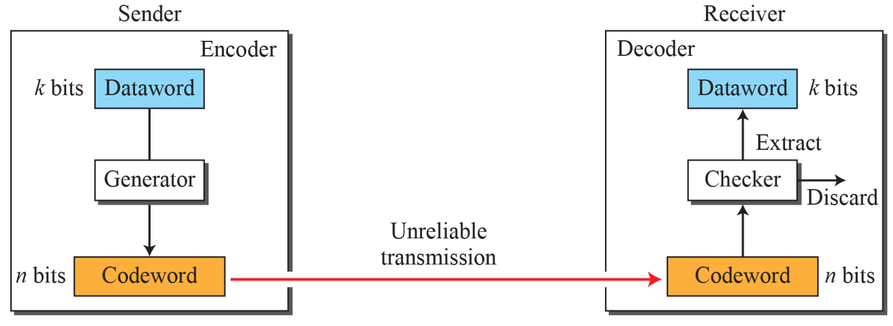


Error Detection in Block Coding



If the codeword is corrupted during transmission but the received word still matches a valid codeword, the **error remains undetected**.

分组编码中的错误检测



如果码字在传输过程中被损坏，但接收到的词仍然匹配一个有效的码字，则**错误未被检测到**。

Hamming Distance

- **Definition**: The **Hamming distance** between two words (of the same size) is the number of differences between corresponding bits.
  - $d(x, y)$: Hamming distance between two words x and y as $d(x, y)$.
  - e.g., the Hamming distance between 0**1**1**0**11 and 0**0**1**1**11 is **two**.

汉明距离

- **定义**：**汉明距离** 指两个等长单词之间对应位不同的数量。
  - $d(x, y)$：两个单词 x 和 y 之间的汉明距离表示为 $d(x, y)$。
  - 例如，0**11**0**1**1 与 0**0**1**11**1 之间的汉明距离为 **二**。

Hamming Distance

- **Definition**: The **Hamming distance** between two words (of the same size) is the number of differences between corresponding bits.
  - $d(x, y)$: Hamming distance between two words x and y as $d(x, y)$.
  - e.g., the Hamming distance between 0**1**1**0**11 and 0**0**1**1**11 is **two**.

- How does the Hamming distance help in error detection?
  - The Hamming distance between the **received codeword** and the **sent codeword** is the **number of bits** that are **corrupted** during transmission.
  - $d(x, y) = 0$ → no error
  - $d(x, y) > 0$ → error

汉明距离

- **定义**：两个（相同长度）单词之间的 **汉明距离** 是指对应位上不同比特的数量。
  - $d(x, y)$：两个单词 x 和 y 之间的汉明距离表示为 $d(x, y)$。
  - 例如，0**1**1**0**11 和 0**0**1**1**11 之间的汉明距离为 **二**。

- 汉明距离如何帮助检测错误?
  - 接收码字 **与** 发送码字 **之间的汉明距离是传输过程中** 发生错误的 **比特数量**，也就是被 **损坏**的比特数。
  - $d(x, y) = 0$ → 无错误
  - $d(x, y) > 0$ → 错误

Hamming Distance (Cont.)

- **Calculating Hamming distance**
  - XOR operation (⊕) on the two words and **count the number of 1s** in the result.
  - Example: $d(10101, 11110) = 3 \leftarrow 10101 \oplus 11110 = 01011$

汉明距离（续）

- **计算汉明距离**
  - 对两个字进行异或操作（⊕），然后**统计结果中1的个数**。
  - 示例：$d(10101, 11110) = 3 \leftarrow 10101 \oplus 11110 = 01011$

Hamming Distance (Cont.)

To **guarantee** the **detection of up to *s* errors** in all cases in a block coding scheme, the **minimum Hamming distance** between all pairs of valid codewords must be $d_{min} = s + 1$.

汉明距离（续）

为了**保证**在所有情况下检测到块中最多**出现*s*个错误**的能力编码方案中，所有有效码字对之间的**最小汉明距离**应为有效码字必须是$d_{min} = s + 1$。

Linear Block Codes

- Almost all block codes used today belong to a subset of block codes called **linear block codes**.
- The use of nonlinear block codes for error detection and correction is **not as widespread** because their structure makes theoretical analysis and implementation difficult.
- The formal definition of linear block codes requires the knowledge of **abstract algebra** (particularly **Galois fields**). → beyond our scope
- **Informal Definition for our purpose**: a **linear block code** is a code in which the **XOR** (addition modulo-2) of two valid codewords creates another valid codeword.

线性分组码

- 如今使用的几乎所有分组码都属于一种称为**线性分组码**的分组码子集。

- 非线性分组码在错误检测和纠正中的应用**并不广泛**，因为其结构使得理论分析和实现较为困难。

- 线性分组码的正式定义需要了解**抽象代数**（特别是**伽罗瓦域**）。→ 超出我们的讨论范围
- **出于本课程目的的非正式定义**：所谓**线性分组码**，是指其中两个有效码字进行**异或**（模2加法）运算后仍产生另一个有效码字的编码。

Linear Block Codes – Parity-Check Code

- **Parity-check code** is a **linear block code**.
- A *k*-bit dataword is changed to an *n*-bit codeword where ***n* = *k* + 1**.
- **Parity bit**
  - The extra bit and it is selected such that the total number of 1s in the codeword becomes **even** (We only consider even parity here, while some implementations specify an odd number of 1s).

线性分组码——奇偶校验码

- **奇偶校验码** 是一种 **线性分组码**。
- 一个 *k* 位的数据字被转换为一个 *n* 位的码字，其中 ***n* = *k* + 1**。
- **奇偶校验位**
  - 额外的一位，其取值选择使得码字中 1 的总数为 **偶数**（此处我们仅考虑偶校验，而某些实现可能规定 1 的个数为奇数）。

Linear Block Codes – Parity-Check Code

- **Parity-check code** is a **linear block code**.
- A *k*-bit dataword is changed to an *n*-bit codeword where ***n* = *k* + 1**.
- **Parity bit**
  - The extra bit and it is selected such that the total number of 1s in the codeword becomes **even** (We only consider even parity here, while some implementations specify an odd number of 1s).
- The minimum Hamming distance for this category is $d_{min} = 2$, which means that the code is **a single-bit error-detecting code**.

线性分组码——奇偶校验码

- **奇偶校验码**是一种**线性分组码**。
- 一个*k*比特的数据字被转换为一个*n*比特的码字，其中***n* = *k* + 1**。
- **奇偶校验位**
  - 该额外比特被选择为使得码字中1的总数为**偶数**（此处仅考虑偶校验，而某些实现可能规定1的个数为奇数）。
- 此类编码的最小汉明距离为$d_{min} = 2$，这意味着该编码是**一种单比特错误检测码**。

Linear Block Codes – Parity-Check Code

- The simple parity check, guaranteed to **detect one single error**, and can also **find any odd number of errors**.
  - E.g., the dataword is **10001** and the codeword **100010** is generated using **even parity bit** of **0** (added to the dataword as the rightmost bit). This codeword is transferred to another computer. In transit, the data is corrupted, and the computer receives the incorrect data **011010**. This codeword has **odd parity** (the number of 1s = 3), therefore, the codeword is **corrupted**.

线性分组码——奇偶校验码

- 简单的奇偶校验，保证能够**检测单个错误**，并且还可以**发现任意奇数个错误**。
  - 例如，数据字为**10001**，使用**100010**作为码字，该码字通过添加**偶校验位**的**0**生成（作为最右边的位添加到数据字中）。此码字被传输到另一台计算机。在传输过程中，数据发生错误，接收端计算机收到错误的数据**011010**。该码字具有**奇校验**（1 的数量为= 3），因此，该码字已**损坏**。

Simple Parity-Check Code – C(5, 4)

- *n*=5, *k*=4 (even parity)

Dataword	*Codeword*	*Dataword*	*Codeword*
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**

简单奇偶校验码 – C(5, 4)

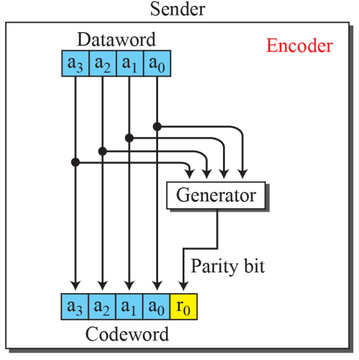
- *n*=5，*k*=4 （偶校验）

Dataword	*Codeword*	*Dataword*	*Codeword*
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**

Encoder and Decoder for Simple Parity-Check Code

- The calculation is done in **modular arithmetic**.

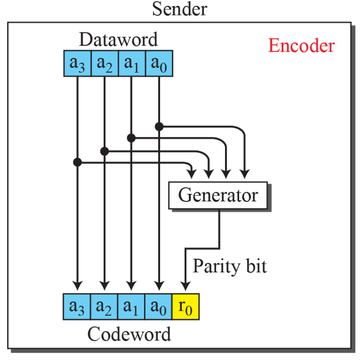
$r_0 = a_3 + a_2 + a_1 + a_0$ (modulo-2)



简单奇偶校验码的编码器和解码器

- 计算在**模运算**中进行。

$r_0 = a_3 + a_2 + a_1 + a_0$(模-2)

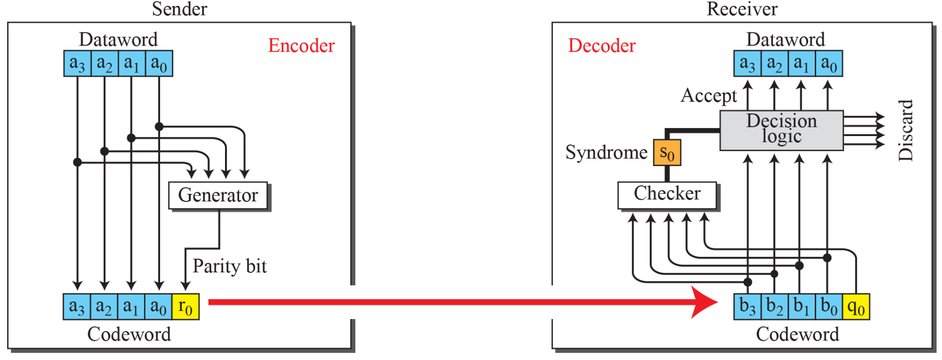


Encoder and Decoder for Simple Parity-Check Code

- The calculation is done in **modular arithmetic**.

$r_0 = a_3 + a_2 + a_1 + a_0$ (modulo-2)

$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$ (modulo-2)

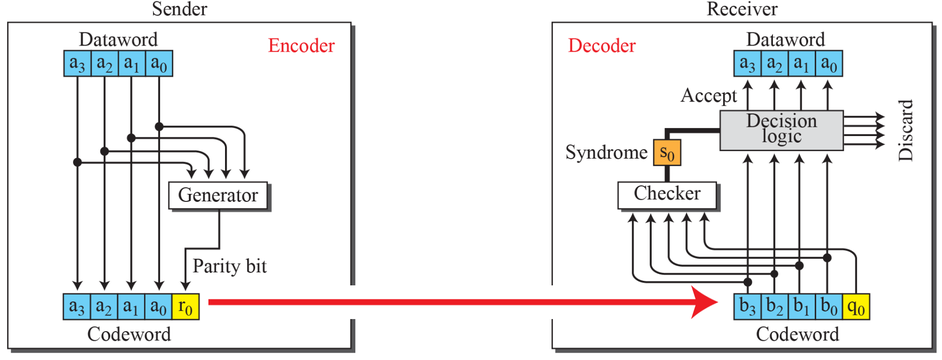


简单奇偶校验码的编码器与解码器

- 计算采用 **模运算**。

$r_0 = a_3 + a_2 + a_1 + a_0$ (modulo-2)

$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$(模-2)

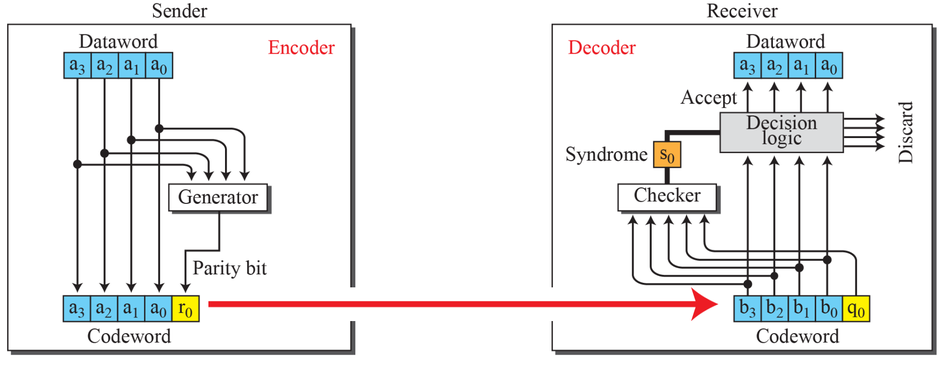


Encoder and Decoder for Simple Parity-Check Code

- The calculation is done in **modular arithmetic**.

$r_0 = a_3 + a_2 + a_1 + a_0$ (modulo-2)

$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$ (modulo-2)



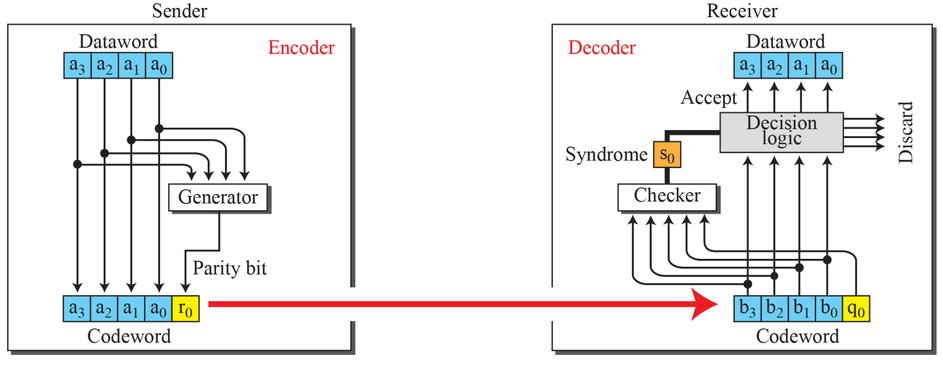
If the syndrome is 0, there is no detectable error in the received codeword.

简单奇偶校验码的编码器和解码器

- 计算采用 **模运算**。

$r_0 = a_3 + a_2 + a_1 + a_0$ (modulo-2)

$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$(模-2)



如果伴随式为0，则接收到的码字中没有可检测到的错误。

Linear Block Codes – Cyclic Codes

- **Cyclic codes**
  - Special linear block codes with **one extra property**: if a codeword is cyclically shifted (**rotated**), the result is another codeword
  - E.g., If **1011000** is a codeword → cyclic left-shift → **0110001** is also a codeword

线性分组码——循环码

- **循环码**
  - 具有 **一个额外性质**的特殊线性分组码：若码字进行循环移位（**旋转**），结果仍是另一个码字
  - 例如，如果 **1011000** 是一个码字，则 → 循环左移 → **0110001** 也是一个码字

Linear Block Codes – Cyclic Codes

- **Cyclic codes**
  - Special linear block codes with **one extra property**: if a codeword is cyclically shifted (**rotated**), the result is another codeword
  - E.g., If **1011000** is a codeword → cyclic left-shift → **0110001** is also a codeword
- **Advantages** of cyclic codes
  - Very good performance in detecting single-bit errors, double errors, and burst errors.
  - Can easily be implemented in hardware and software.
  - Especially fast when implemented in hardware.

线性分组码——循环码

- **循环码**
  - 具有**一个额外性质**的特殊线性分组码：若码字进行循环移位（**旋转**），结果仍是另一个码字
  - 例如，如果 **1011000** 是一个码字，→ 循环左移 → **0110001** 也同样是码字
- **循环码的优点**
  - 在检测单比特错误、双比特错误和突发错误方面性能非常出色。
  - 可以轻松地在硬件和软件中实现。
  - 在硬件中实现时尤其快速。

Linear Block Codes – Cyclic Codes

- **Cyclic codes**
  - Special linear block codes with **one extra property**: if a codeword is cyclically shifted (**rotated**), the result is another codeword
  - E.g., If **1011000** is a codeword → cyclic left-shift → **0110001** is also a codeword
- **Advantages** of cyclic codes
  - Very good performance in detecting single-bit errors, double errors, and burst errors.
  - Can easily be implemented in hardware and software.
  - Especially fast when implemented in hardware.
- **Our focus**
  - A subset of cyclic codes called **Cyclic Redundancy Check** (**CRC**), which is used in LANs and WANs

线性分组码——循环码

- **循环码**
  - 具有**一个额外性质**的特殊线性分组码：若码字进行循环移位（**旋转**），结果仍是另一个码字
  - 例如，如果 **1011000** 是一个码字，→ 循环左移 → **0110001** 也同样是码字
- **循环码的优点**
  - 在检测单比特错误、双比特错误以及突发错误方面性能非常出色。
  - 可以轻松地在硬件和软件中实现。
  - 在硬件中实现时尤其快速。
- **我们的重点**
  - 一类称为**循环冗余校验**（**CRC**），用于局域网和广域网

A CRC Code with C(7, 4)

- *n*=7, *k*=4

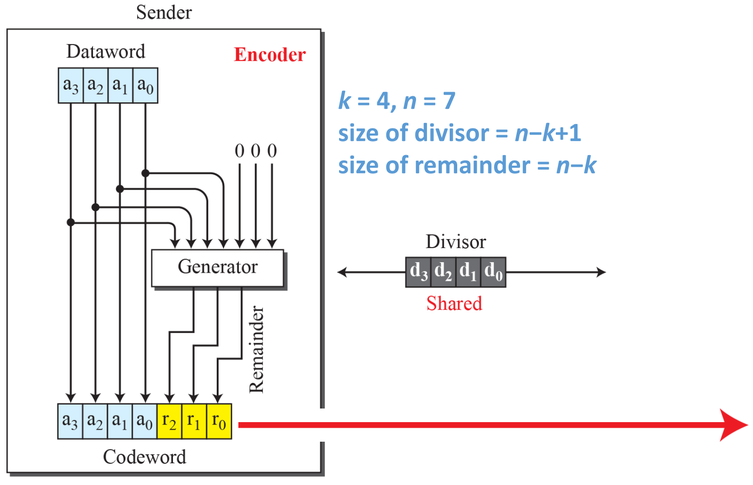
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

一个C(7, 4)的CRC码

- n=7，k=4

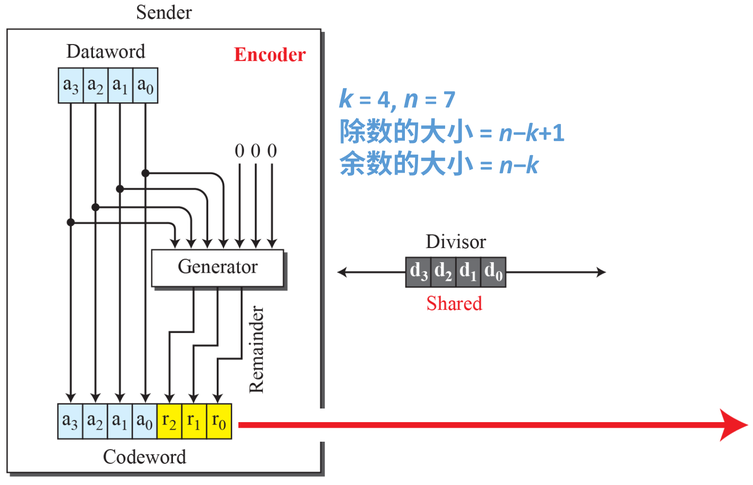
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

CRC Encoder and Decoder



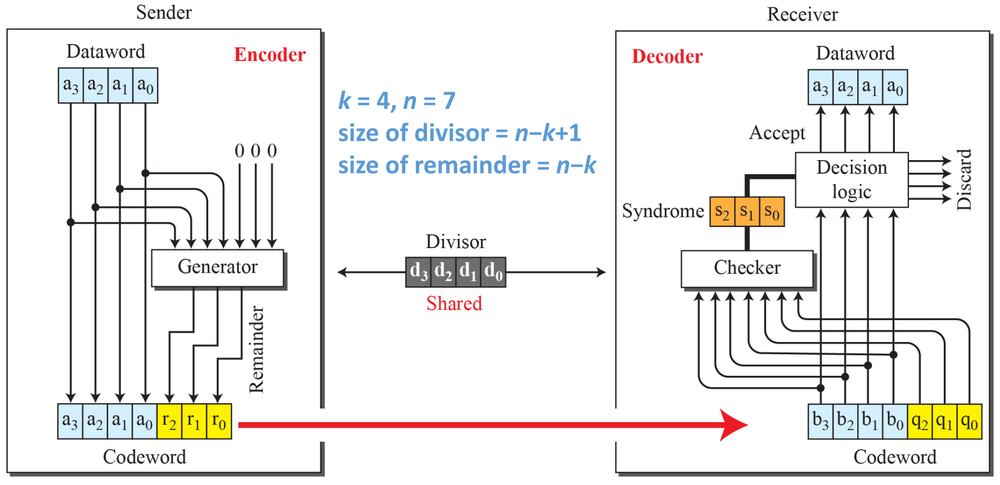
k = 4, *n* = 7
size of divisor = *n*–*k*+1
size of remainder = *n*–*k*

CRC编码器和解码器

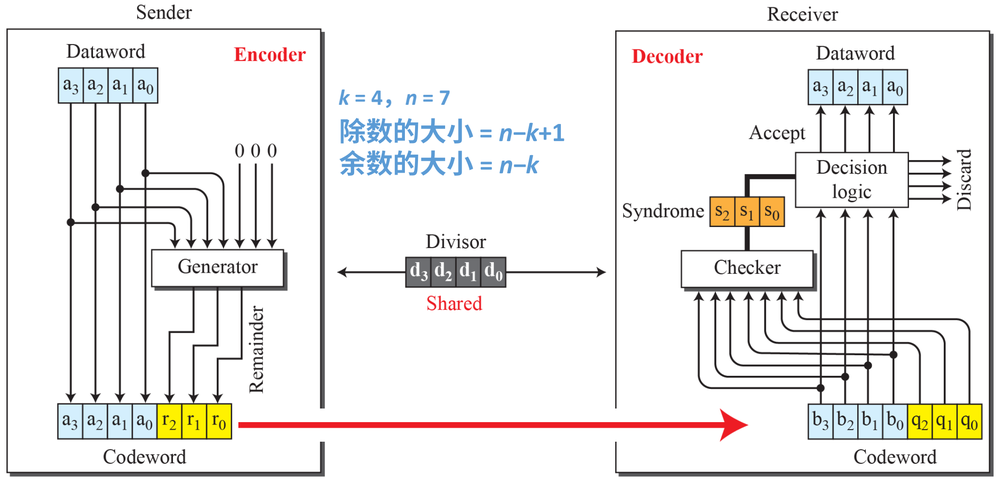


k = 4, *n* = 7
除数的大小 = *n*–*k*+1
余数的大小 = *n*–*k*

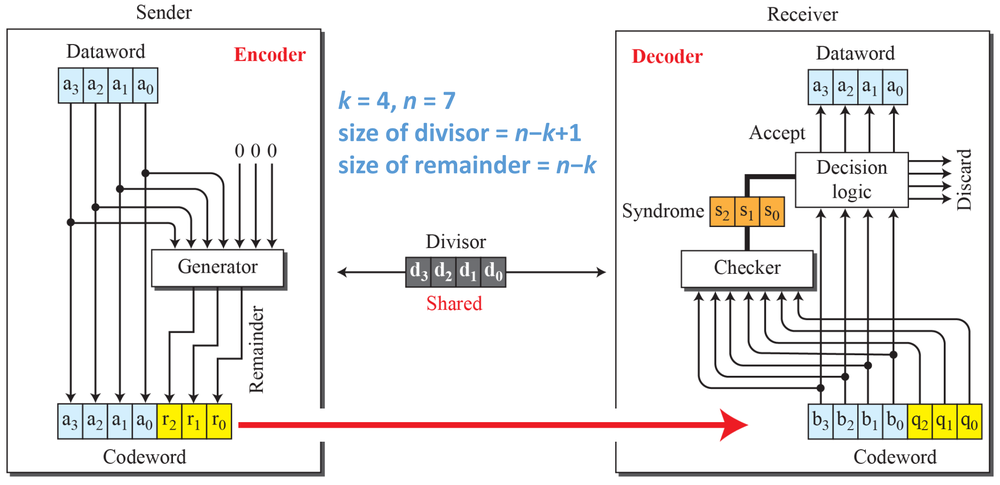
CRC Encoder and Decoder



CRC编码器和解码器

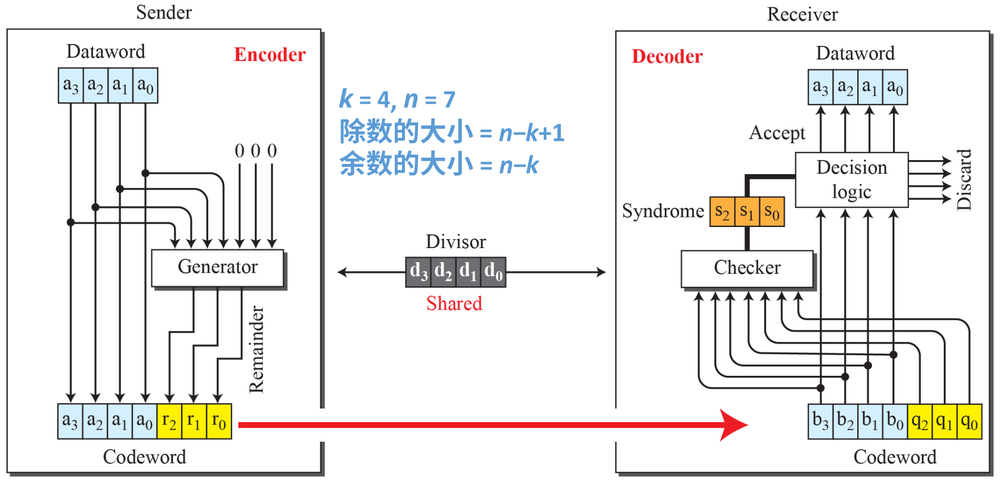


CRC Encoder and Decoder



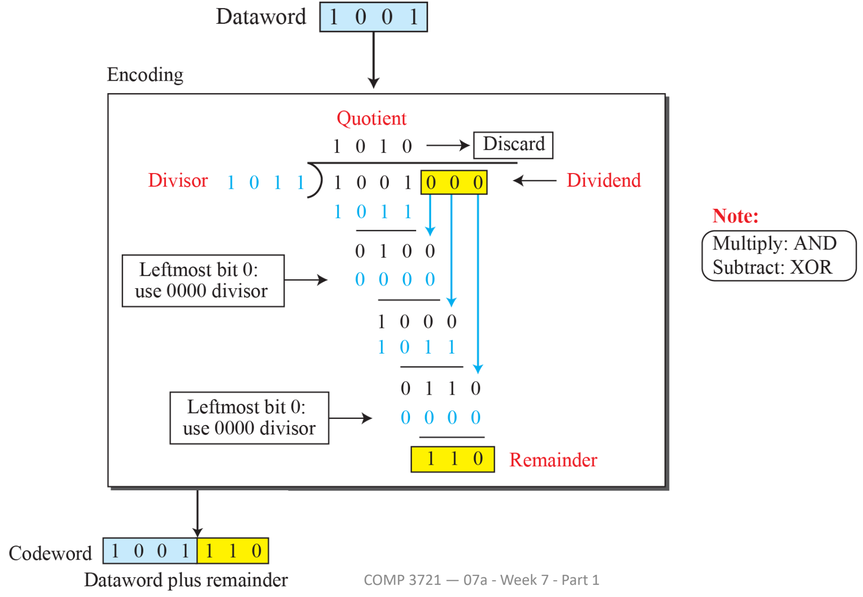
If syndrome is 0, either no bit is corrupted, or the decoder failed to detect any errors.

CRC编码器和解码器

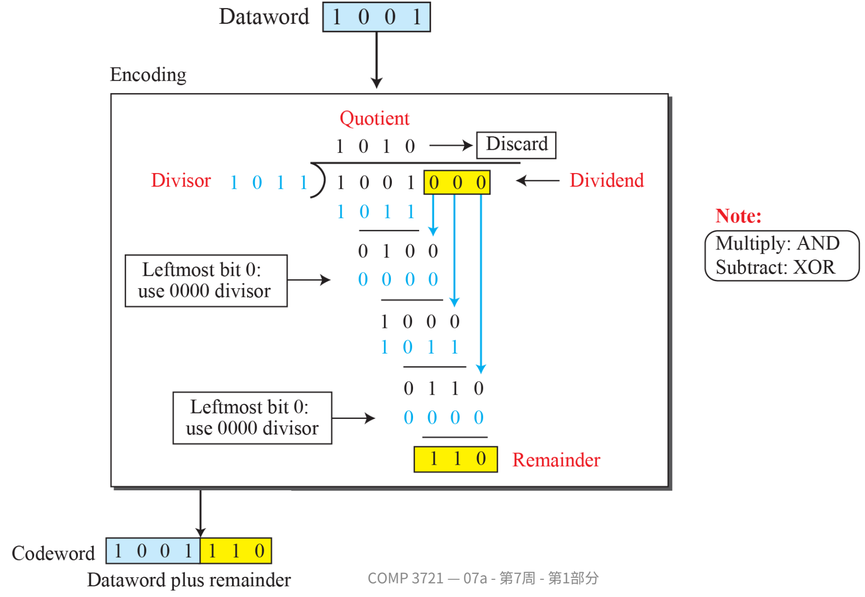


如果伴随式为0，则要么没有比特出错，要么解码器未能检测到任何错误。

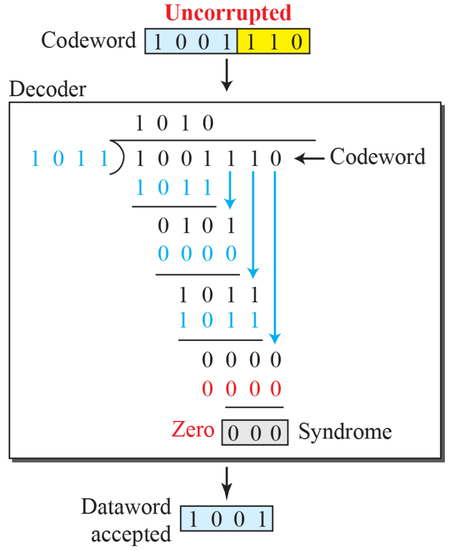
Division in CRC Encoder



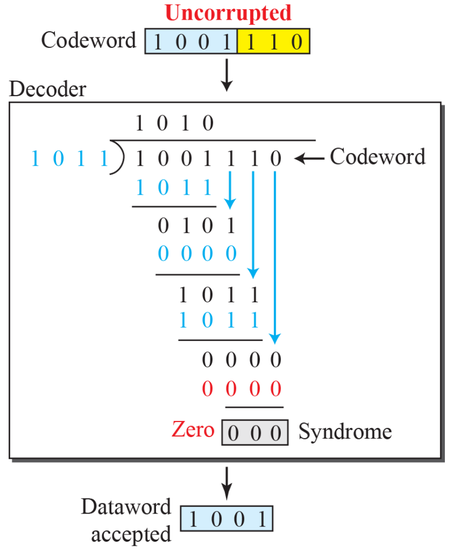
CRC编码器中的除法



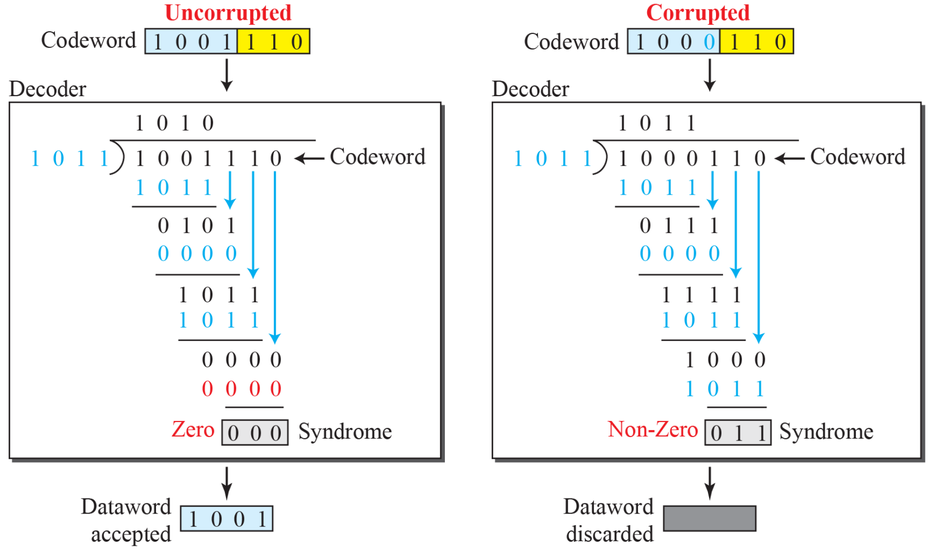
Division in CRC Decoder for Two Cases



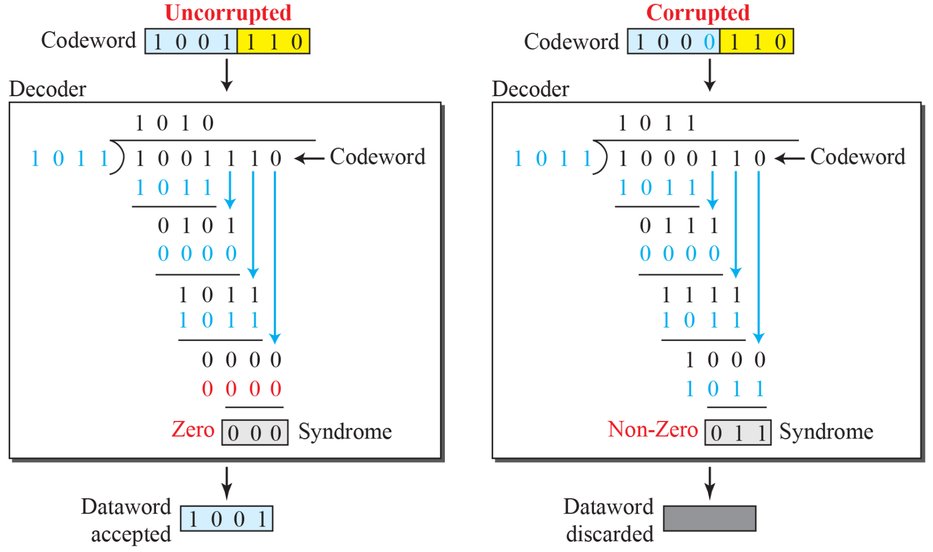
两种情况下的CRC解码器中的除法



Division in CRC Decoder for Two Cases



两种情况下的CRC解码器中的除法



Checksum

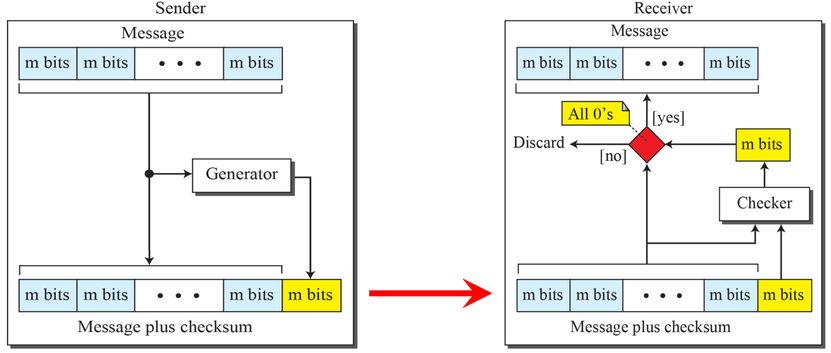
- An **error-detecting technique** that can be applied to a message of **any length**.
- Mostly used at the **network** and **transport layers** rather than the data-link layer.

校验和

- 一种**错误检测技术**，可应用于**任意长度**的消息。
- 主要用在**网络层**和**传输层**，而不是数据链路层。

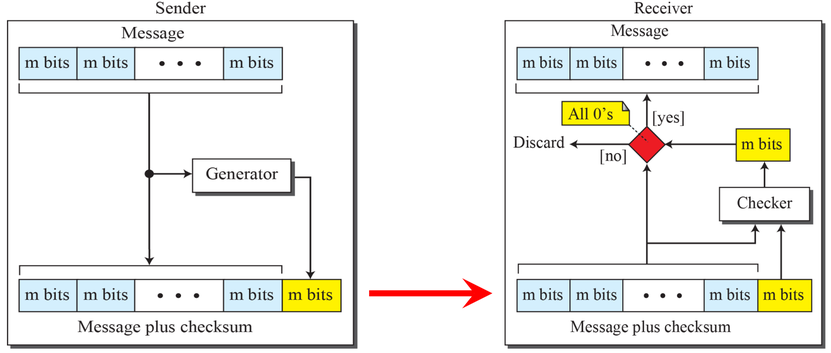
Checksum

- An **error-detecting technique** that can be applied to a message of **any length**.
- Mostly used at the **network** and **transport layers** rather than the data-link layer.



校验和

- 一种**错误检测技术**，可应用于**任意长度**的消息。
- 主要用在**网络层**和**传输层**，而非数据链路层。



Checksum – Example

- Suppose the message is a list of **five 4-bit numbers** that we want to send to a destination (**7**, **11**, **12**, **0**, **6**):
  - In addition to sending these numbers, we send the **sum** of the numbers.
  - E.g., we send (**7**, **11**, **12**, **0**, **6**, **36**), where **36** is the sum of the original numbers.
  - The receiver adds the five numbers and compares the result with the sum.
  - If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum.
  - Otherwise, there is an error somewhere and the message is not accepted.

校验和——示例

- 假设消息是要发送到目的地的一组 **五个4位数字**（**7**，**11**，**12**，**0**，**6**）：
  - 除了发送这些数字外，我们还发送这些数字的 **和**。
  - 例如，我们发送（**7**，**11**，**12**，**0**，**6**，**36**），其中 **36** 是原始数字之和。
  - 接收方将这五个数字相加，并将结果与所发送的和进行比较。
  - 如果两者相同，接收方认为没有错误，接受这五个数字，并丢弃该和。
  - 否则，某处存在错误，消息将不被接受。

Checksum – Example

- The previous example has one issue:
  - Each number can be written as a 4-bit word (each is less than 15) **except for the sum**.

- **Solution**: Use **one's complement** arithmetic.

校验和——示例

- 前面的示例存在一个问题：
  - 每个数字都可以表示为一个4位字（每个都小于15）**但总和除外**。

- **解决方案**：使用**一补码**算术。

Checksum – One's Complement Arithmetic

- Unsigned numbers between 0 and 2^m-1 are represented using only m bits.

校验和——一的补码算术

- 介于 0 和 2_{m-1} 之间的无符号数使用仅 m 位表示。

Checksum – One's Complement Arithmetic

- Unsigned numbers between 0 and 2^m-1 are represented using only *m* bits.
- If the number has more than *m* bits, the **extra leftmost bits** need to be added to the ***m* rightmost bits** (**wrapping**).
  - E.g., decimal number 36 in binary → **100100**, to change it to a 4-bit number: $(\mathbf{10})_2 + (\mathbf{0100})_2 = (0110)_2 \rightarrow (6)_{10}$

校验和——反码算术

- 介于 0 和 2_{m-1} 之间的无符号数仅用*m*位表示。
- 如果数字的位数超过 *m* 位，则最左边的 **多余位** 需要加到最右边的 ***m* 位上**（**回卷**）。
  - 例如，十进制数 36 的二进制形式为 → **100100**，将其转换为 4 位数：$(\mathbf{10})_2 + (\mathbf{0100})_2 = (0110)_2 \rightarrow (6)_{10}$

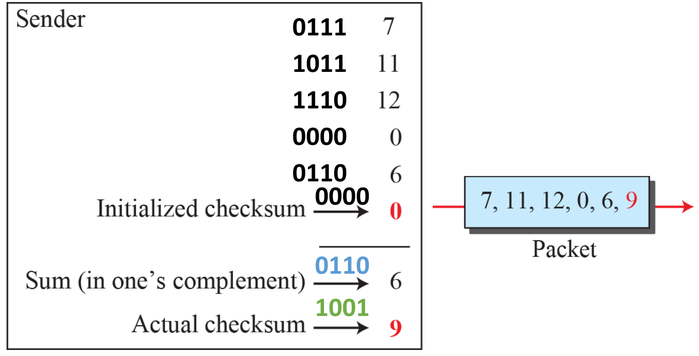
Checksum – One's Complement Arithmetic

- Unsigned numbers between 0 and 2^m-1 are represented using only *m* bits.
- If the number has more than *m* bits, the **extra leftmost bits** need to be added to the ***m* rightmost bits** (**wrapping**).
  - E.g., decimal number 36 in binary → **100100**, to change it to a 4-bit number: $(\mathbf{10})_2 + (\mathbf{0100})_2 = (0110)_2 \rightarrow (6)_{10}$
- The **complement** of a number is found by changing all 1s to 0s and all 0s to 1s.
  - The same as subtracting the number from 2^m-1.
- In one's complement arithmetic, we have **two 0s**: one positive and one negative, which are complements of each other.
- The positive zero has all *m* bits set to 0 (**0000**); the negative zero has all bits set to 1 (it is 2^m-1) (**1111**).

校验和——反码算术

- 介于 0 和 2_{m-1} 之间的无符号数使用仅含 *m* 位的表示方式。
- 如果该数字的位数超过 *m* 位，则需要将最左侧的 **多余位** 加到最右侧的 ***m* 位上**（**回卷**）。
  - 例如，十进制数 36 的二进制形式为 → **100100**，将其转换为 4 位数：$(\mathbf{10})_2 + (\mathbf{0100})_2 = (0110)_2 \rightarrow (6)_{10}$
- 通过将所有的 1 变为 0，所有的 0 变为 1，即可得到一个数的 **反码**。
  - 与从2_m-1中减去该数相同。
- 在反码运算中，我们有**两个0**：一个为正，一个为负，它们互为补码。
- 正零的所有*m*位都为0（**0000**）；负零的所有位都为1（即2_m-1）（**1111**）。

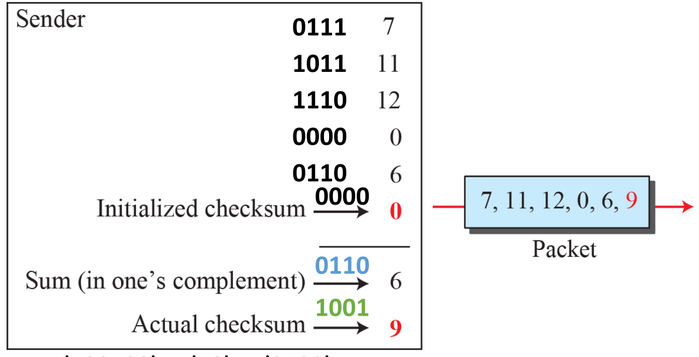
Checksum – Example (4-bit binary numbers)



36 = $(100100)_2$ = $(10)_2$ + $(0100)_2$
= $(0110)_2$ = 6

One's Complement of 6 is 9 = $(1001)_2$

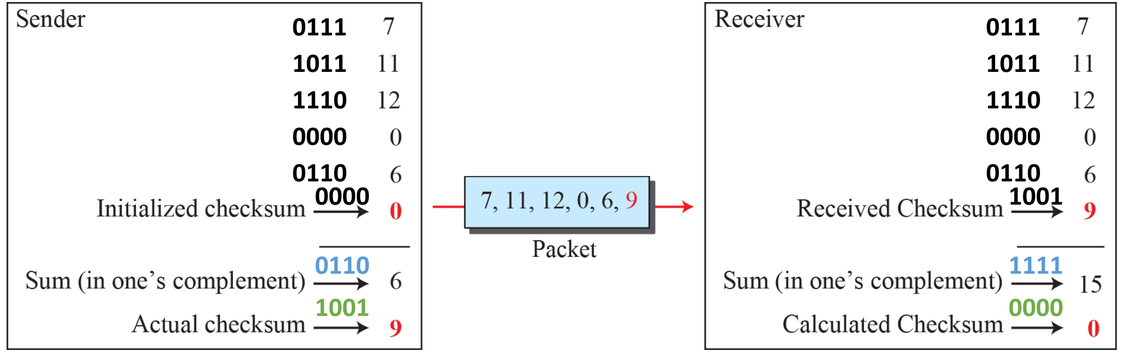
校验和——示例（4位二进制数）



36 = $(100100)_2$ = $(10)_2$ + $(0100)_2$
= $(0110)_2$ = 6

6的反码是6是9 = $(1001)_2$

Checksum – Example (4-bit binary numbers)



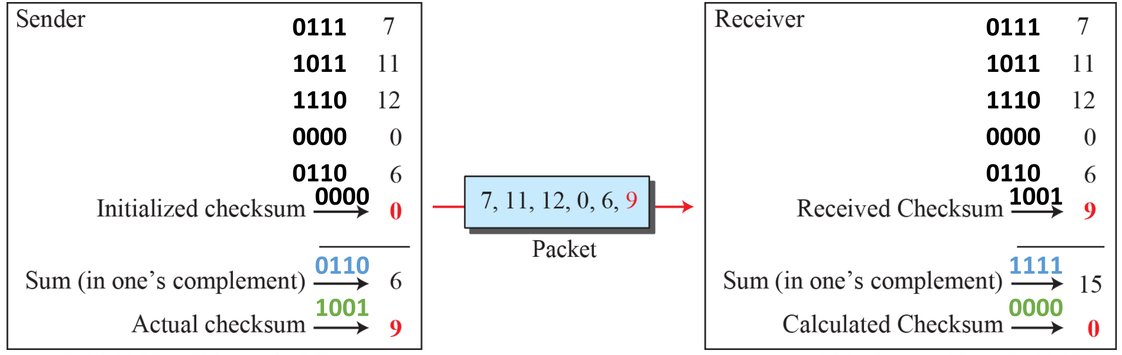
36 = $(100100)_2$ = $(10)_2$ + $(0100)_2$
= $(0110)_2$ = 6

One's Complement of 6 is 9 = $(1001)_2$

36 + 9 = 45 = $(101101)_2$ = $(10)_2$ + $(1101)_2$
= $(1111)_2$ = 15

One's Complement of 15 is 0 = $(0000)_2$

校验和——示例（4位二进制数）



36 = $(100100)_2$ = $(10)_2$ + $(0100)_2$
= $(0110)_2$ = 6

6 的反码是 6 的反码为 9 = $(1001)_2$

36 + 9 = 45 = $(101101)_2$ = $(10)_2$ + $(1101)_2$
= $(1111)_2$ = 15

15 的原码为 15 是 0 = （$(0000)_2$

Procedure to Calculate the Traditional Checksum

- Traditionally, the Internet has used a 16-bit checksum.

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

传统校验和的计算步骤

- 传统上，互联网使用的是16位校验和。

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Forward Error Correction (FEC)

- Retransmission of corrupted and lost packets is not useful for real-time **multimedia transmission**.
  - Due to **unacceptable delay** in reproducing.
- In such applications, we need to correct the error or reproduce the packet immediately.

前向纠错（FEC）

- 对于实时**多媒体传输**，重传损坏和丢失的数据包是无用的。
  - 由于**播放时延迟不可接受**。
- 在这些应用中，我们需要立即纠正错误或重建数据包。

FEC Techniques

- Hamming distance
- Using XOR
- Chunk interleaving
- …

FEC 技术

- 汉明距离
- 使用异或运算
- 分块交织
- …

FEC Techniques – Hamming Distance

- To **detect *s* errors**, $d_{min} = s + 1$. For error correction, we need more distance.
- To **correct *t* errors**, we need to have $d_{min} = 2t + 1$.
  - E.g., if we want to correct 10 bits in a packet, we need to make the minimum hamming distance 21 bits → lot of redundant bits need to be sent with the data

前向纠错技术——汉明距离

- 要**检测 *s* 个错误**，$d_{min} = s + 1$。对于纠错，我们需要更多距离。
- 要**纠正 *t* 个错误**，我们需要有 $d_{min} = 2t + 1$。
  - 例如，如果我们想在一个数据包中纠正 10 个比特的错误，就需要使最小汉明距离达到 21 个比特，→ 需要随数据发送大量冗余比特

FEC Techniques – Using XOR

- Using the property of XOR operation:

$$\mathbf{R} = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N$$

- This means:

$$\mathbf{P}_i = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

FEC 技术——使用异或运算

- 利用异或运算的性质：

$$\mathbf{R} = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N$$

- 这意味着：

$$\mathbf{P}_i = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

FEC Techniques – Using XOR

- Using the property of XOR operation:

$$\mathbf{R} = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N$$

- This means:

$$\mathbf{P}_i = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

- We can divide a packet into *N* chunks, create the XOR of all the chunks and send *N*+1 Chunks. If any chunk is lost or corrupted, it can be created at the receiver site.
- (If *N* = 4, it means that we need to send 25 percent extra data and be able to correct the data if only one out of four chunks is lost.)

FEC 技术——使用异或运算

- 利用异或运算的特性：

$$\mathbf{R} = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N$$

- 这意味着：

$$\mathbf{P}_i = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

- 我们可以将一个数据包分成*N*个块，计算所有块的异或值，并发送*N*+1 个块。如果任何一个块丢失或损坏，接收端都可以重新生成该块。
- （如果*N* = 4，意味着我们需要多发送25%的额外数据，并且在每四个块中仅丢失一个块的情况下能够纠正数据。）

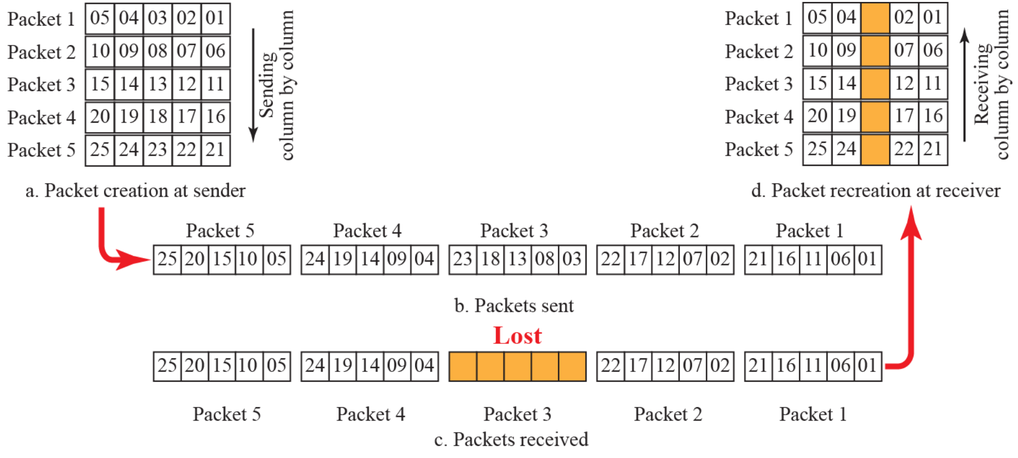
FEC Techniques – Chunk Interleaving

- Allowing some small chunks to be missing at the receiver.
  - Not all the chunks belonging to the same packet can be missing.
  - We can afford to let one chunk be missing in each packet.

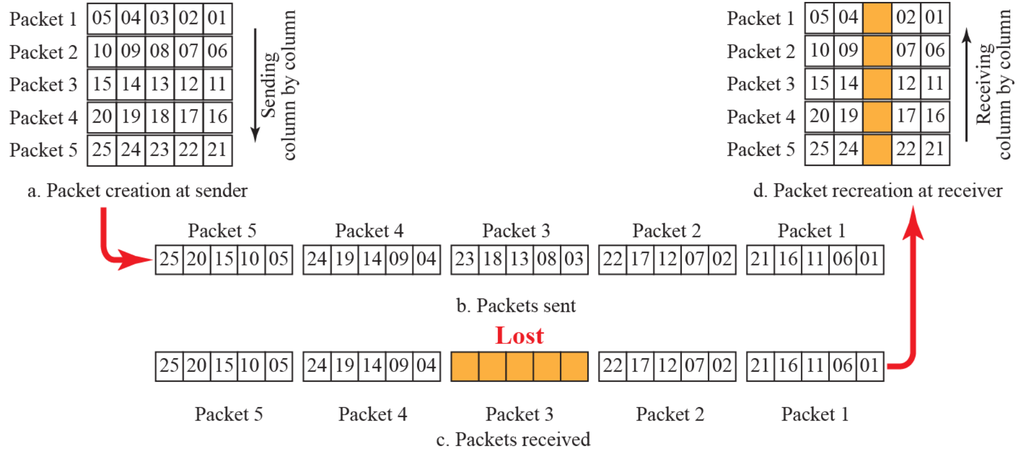
FEC 技术——分块交织

- 允许接收端丢失少量分块。
  - 属于同一数据包的所有分块不能全部丢失。
  - 我们可以承受每个数据包中有一个分块丢失。

Chunk Interleaving – Example



块交错——示例



Summary

- Possibility of data corruption during transmission
- Sending redundant bits with data for error detection/correction
- Block coding
- Hamming distance
- Cyclic codes as special linear block codes
- Internet checksum to detect errors in messages of any size

摘要

- 数据传输过程中可能发生数据损坏
- 发送带有冗余比特的数据以进行错误检测/纠正
- 分组编码
- 汉明距离
- 循环码作为特殊的线性分组码
- 用于检测任意大小消息中错误的互联网校验和

References

[1] Behrouz A.Forouzan, Data Communications & Networking with TCP/IP Protocol Suite, 6th Ed, 2022, McGraw-Hill companies.

参考文献

[1] Behrouz A.Forouzan, Data Communications & Networking with TCP/IP 协议套件，第6版，2022年，麦格劳-希尔公司。

Reading

- Chapter 3 of the textbook, section 3.2.2.
- Chapter 3 of the textbook, section 3.6 (Practice Test)

阅读

- 教材第3章，第3.2.2节。
- 教材第3章，第3.6节（练习测试）