CRC 校验漏检的充要条件及编程验证

hxjelly 网络空间安全学院 UCAS

摘 要：CRC(Cyclic Redundancy Check)循环冗余校验是根据一段数据产生简短固定位数校验码的一种信道编码技术，利用除法及余数的原理来检测或校验数据传输或者保存后可能出现的错误。CRC 校验计算速度快，检错能力强，易于用编码器等硬件电路实现。然而，CRC 校验只能保证丢掉的数据段一定出现了比特差错，不能保证未丢掉的数据段一定没有比特差错，即 CRC 校验没有错检但存在漏检。本文将首先在原理上理论分析何时会发生漏检并给出充要条件，然后通过 SageMath 语言编程验证结果。

关键词：CRC；差错检测控制；循环冗余校验；漏检率；比特差错

一、CRC 漏检原因分析

在开始分析之前，先对本文中的符号等进行统一说明。

1.1 符号说明与一些假定

一个二进制序列可以写成多项式的形式，序列中各位上的数值就是多项式的系数。例如 6 位二进制序列 $g_5g_4g_3g_2g_1g_0$ 可以写成

$$G(x)=g_5\cdot x^5+g_4\cdot x^4+g_3\cdot x^3+g_2\cdot x^2+g_1\cdot x^1+g_0\cdot x^0$$

其中 $g_i\in\{0,1\}(i=0,1,...,5)$ 。

假设待发送的数据是 k 位二进制序列 $M(x)$ 。将其向左偏移 r 位，右边补 0，可得到 $x^rM(x)$ 。

再给定 $r+1$ 位二进制序列，记对应的多项式为 $P(x)$，称为**生成多项式**。这里为方便说明，假定 $P(x)$ 的首位必须为 1，即 $\deg(P(x))=r$ 。在编程验证时，给定 11001 合法，而给定 01001 不合法。

用 $x^rM(x)$ 作为被除式， $P(x)$ 作为除式，进行带余除法，则有

$$x^rM(x)=Q(x)*P(x)\oplus N(x)$$

其中 $\oplus$ 表示 $\mathbb{F}_2$ 二元域上的加法即按位异或， $Q(x)$ 为商， $N(x)$ 为余式，满足

$$\deg(N(x))<\deg(P(x))=r$$

即 $N(x)$ 对应的比特序列最多只有 r 位。

定义 $T(x)=x^rM(x)\oplus N(x)$ （其中 $\oplus$ 表示按位异或），则 $T(x)$ 将是一个 n 位

二进制序列，$n=k+r$。低 r 位称为 $T(x)$ 的 **CRC 校验值**，高 k 位称为**有效数据**。

后面的分析均基于这些符号进行。即有效数据比特位数为 k，CRC 校验值比特位数为 r，总比特位数为 n。同时需要注意我们的所有运算都是在 $\mathbb{F}_2$ 即 2 元域下进行，例如加减乘除等。

1.2 CRC 的基本原理与检验过程

下面简要介绍 CRC 的基本原理与接收方的检验过程。

发送方将 $T(x)=x^rM(x)\oplus N(x)$ 发送至接收方。设接收方收到的多项式为 $R(x)$。由于通信过程中的各种干扰，$R(x)$ 可能与 $T(x)$ 不同。为了验证是否存在比特错误，接收方将 $R(x)$ 作为被除式，$P(x)$ 作为除式进行带余除法。若余式不为 0，则说明一定发生了比特错误，然后接收方可以要求发送方重新发送数据；若余式为 0，则说明 $R(x)$ 与 $T(x)$ 相同的概率很大，但也可能不同。

在教科书中并没有进一步分析漏检的发生原因，而本部分接下来将着重探讨漏检发生的原因与条件。

1.3 CRC 漏检相关分析

首先先明确什么情况属于 CRC 漏检。沿用前两小节的符号，发送方将 $T(x)=x^rM(x)\oplus N(x)$ 发送至接收方，而接收方收到的多项式为 $R(x)$。称此时**发生了 CRC 漏检**，如果 $R(x)\neq T(x)$ 但 $P(x)\mid R(x)$。

那么给定 $T(x)$、$R(x)$ 和生成多项式 $P(x)$，能否以概率 1 确定此时是否发生 CRC 漏检呢？答案是肯定的。我们给出如下结论：

给定 $T(x)$、$R(x)$（$T(x)\neq R(x)$）和生成多项式 $P(x)$，设 $E(x)=T(x)\oplus R(x)$，则此时**发生 CRC 漏检的充要条件是** $P(x)\mid E(x)$。

证明：

先证明一个结论：$P(x)\mid T(x)$。

事实上，由于

$$x^rM(x)=Q(x)*P(x)\oplus N(x)$$

那么根据异或性质，有

$$Q(x)*P(x)=x^rM(x)\oplus N(x)=T(x)$$

即

$$T(x)=Q(x)*P(x)$$

故 $P(x)\mid T(x)$ 成立，证毕。

然后证明正文给出的结论：

① 必要性。由 CRC 漏检的定义可得，此时 $P(x)\mid R(x)$。由于 $P(x)\mid T(x)$，故

由整除性质可得 $P(x)|T(x)\oplus R(x)$，即 $P(x)|E(x)$，必要性得证。

② 充分性。由 $P(x)|E(x)$ 和 $P(x)|T(x)$，以及整除性质，可得 $P(x)|E(x)\oplus T(x)$，即 $P(x)|T(x)\oplus R(x)\oplus T(x)$，即 $P(x)|R(x)$。此时 $P(x)|R(x)$ 并且 $T(x)\neq R(x)$，满足 CRC 漏检之定义，充分性得证。

必要性和充分性均得证，故结论证毕。

对这个充要条件稍加解释。如果将引入的多项式 $E(x)=T(x)\oplus R(x)$ 表示成比特序列，那么其含义便比较直观：比特位为 1 则表示发送方发送的多项式 $T(x)$ 与接收方实际接收的多项式 $R(x)$ 在该位上发生了比特差错；为 0 则表示该位在传输时没有比特差错。这样我们便得到了一个发生 CRC 漏检的充要条件，并作为后面编程验证的核心依据。

例 1：生成多项式

$$P(x)=11001$$ [1]

发送方发送的有效数据

$$M(x)=110011$$

参数 n=10，k=6，r=4。带余除法计算可得

$$N(x)=1001$$

发送端最终发送的多项式为

$$T(x)=x^4M(x)\oplus N(x)=1100111001$$

在传输没有错误的情况下，接收端接收的信息比特序列 $R(x)$ 也应该是 1100111001。但是如果从左数第 6、第 7 和第 10 位发生了比特翻转，即

$$R(x)=1100100000$$

那么此时 $R(x)=T(x)\oplus P(x)$。而 $T(x)=Q(x)*P(x)$，故 $P(x)|R(x)$，发生了 CRC 漏检，即 $R(x)$ 仍然可以通过 CRC 校验。

根据本文提出的充要条件来验证也是可以得到此结论的。经计算可得此时 $E(x)=T(x)\oplus R(x)=11001$，$P(x)=11001$，故 $P(x)|E(x)$ 成立，即满足发生 CRC 漏检的充要条件。故 $R(x)$ 虽然错了 3 个比特，但 $R(x)$ 仍然可以通过 CRC 校验。

根据这个充要条件可以发现，只要给定参数 n、k、r，以及生成多项式 $P(x)$，就可以通过计算 $E(x)$ 得到 n 个比特中的哪些比特发生翻转仍然可以通过 CRC 校验，而与发送的数据 $T(x)$ 本身的内容无关。

下面开始实际编程验证。

[1] 注：这样写是表示 $P(x)$ 对应的比特序列为 11001，下同。

二、CRC 漏检的编程验证

在编程之前，首先先明确一下模拟的场景，以及输入和输出。

2.1 场景描述与需求分析

在某个通信质量较差的数据链路上传递 n 个比特的消息，其中高 k 位为有效数据，低 r 位为 CRC 校验值，$r=n-k$。结果有 i 个比特发生比特翻转（即 0 误传为 1，1 误传为 0），但仍能通过 CRC 校验，也就是说被漏检了。

由用户给定生成多项式 $P(x)$，参数 n 和比特翻转个数 i，程序应能给出哪 i 个位置的比特发生翻转后，仍能通过 CRC 校验。

2.2 设计思路

首先，用户给定参数 n 和生成多项式 $P(x)$（要求首位必须为 1）后，便可以通过 $P(x)$ 的位数定出 r 值，进而算出 k 值。

然后核心就是 $E(x)$ 的计算了。因为前文提到过，$E(x)$ 的比特序列中的 1 的位置即表示这 n 个比特的消息传输时发生比特翻转的位置。

我们设计了两个算法方案，并会根据复杂程度（循环的具体次数等）判断选用哪一个。

2.3 方案 1

设 $E(x)=C(x)*P(x)$。由于 $\deg(P(x))=r$，且 $\deg(E(x))\le n-1$，故可以遍历 $C(x)$ 从 0…01 到 01…1[2]，共 2^k-1 种比特序列（对应十进制数即从 1 到 2^k-1）。每次遍历的时候，计算 $E(x)$ 并判断 $E(x)$ 比特序列所含 1 的个数，如果等于 i 则保留这个 $E(x)$ 并放入全局列表 E 中，算法结束；如果不等于 i 则丢弃之，继续进行下一个 $C(x)$ 的遍历。最后，在全局列表 E 中得到了所有的可能 $E(x)$ 后，通过遍历 E 列表，一并地将比特翻转的位数等细节输出至终端即可。

这里对 $C(x)$ 的遍历范围进行简要说明：之所以 $C(x)$ 不能遍历到 10…0（对应十进制数为 2^k），是因为此时 $\deg(C(x))=k$。那么便有

$$\deg(E(x))=\deg(P(x))+\deg(C(x))=r+k=n>n-1$$

不满足 $\deg(E(x))\le n-1$ 的要求。而 $C(x)$ 遍历到 01…1（对应十进制数为 2^k-1）时，$\deg(C(x))=k-1$，此时 $\deg(E(x))=n-1$，满足要求。所以 $C(x)$ 从 0…01 到 01…1 遍历。

方案 1 算法设计的核心用伪代码的形式表示如下：

---

[2] 这里指的是 $C(x)$**的比特序列**对应从 0..01(1 的二进制表示)到 10..0(2^k 的二进制表示)遍历。下文中类似的说法均默认为此含义。

Algorithm 1: 方案 1

Input: 总比特位数 n，生成多项式 $P(x)$ 和期望比特翻转位数 i

Output: 能通过 CRC 校验的所有 i 位比特翻转的情况

```
calculate k and r
for i←1 to 2^k − 1 do
        E(x) = i * P(x)
        count← E(x).coefficients()
        if count == i then
                E.append( E(x) )
        endif
end for
if isEmpty(E) then
        print('No possible case that can pass the CRC!')
else then
        print(str(len(E))+' cases that can pass the CRC:')
        output details of all the cases according to the global list E
endif
return None
```

可以看到此时的循环迭代次数为 2^k-1，是 $O(2^n)$ 的，即指数级增长。

方案 1 的思路是先遍历筛出所有可被 $P(x)$ 整除的 $E(x)$，再通过参数 i 筛选出所有符合要求的 $E(x)$。那么一个自然的新思路便随之产生——先通过参数 i 筛选出 1 比特数为 i 的所有 $E(x)$，再通过整除性筛出所有可被 $P(x)$ 整除的 $E(x)$。此即方案 2。

2.4 方案 2

方案 2 与方案 1 相反，这里先通过组合遍历得到 1 比特数为 i 的所有 $E(x)$，对应的函数为 recur(tot,num)，含义是从 tot 个比特中取 num 个为 1，剩余为 0。将所有满足此要求的比特序列存储到一个列表中并返回。

采用递归来实现。与传统的递归不同的是，这里设置了很多收敛条件（边界条件），以期降低递归层数从而优化性能。

组合遍历结束后，遍历所有的比特序列对应的多项式，如果能被 $P(x)$ 整除则保留这个 $E(x)$ 并放入全局列表 E 中，算法结束；如果不能被 $P(x)$ 整除则丢弃之，继续进行下一个比特序列的遍历。

最后，在全局列表 E 中得到了所有的可能 $E(x)$ 后，通过遍历 E 列表，一并地将比特翻转的位数等细节输出至终端即可。

recur(tot,num)函数用伪代码的形式表示如下：

Algorithm 2: 组合遍历的实现 recur(*tot*,*num*)

Input: *tot* 和 *num*，含义是从 *tot* 个比特中取 *num* 个为 1，剩余为 0

Output: 存储所有满足此要求的比特序列的列表

```
1:  res←[]
2:  #corner cases:
3:  if tot < num then
4:      res.append("null")
5:      return res
6:  endif
7:  if tot == num then
8:      res.append("1…1")   # the number of 1 is num
9:      return res
10: endif
11: if tot == 1 then
12:     if num == 1 then     #1,1
13:         res.append("1")
14:         return res
15:     endif
16:     if num == 0 then
17:         res.append("0")
18:         return res
19:     endif
20:     res.append("null")
21:     return res
22: endif
23: if num == 0 then
24:     res.append("0…0")   # the number of 0 is tot
25:     return res
26: endif
27: if num == 1 then
28:     #the number of 0 is tot-1
29:     res.append("10…0")   # the number of 0 after 1 is tot-1
30:     res.append("010…0")   # the number of 0 after 1 is tot-2
31:     res.append("0010…0")   # the number of 0 after 1 is tot-3
32:     … …
33:     res.append("00…001")   # the number of 0 after 1 is 0
34:     return res
35: endif
36: #not corner cases:
37: a←recur(tot-1,num-1)
38: if "null" is not in a then
39:     for i in a do
40:         res.append("1"+i)
41:     endfor
42: endif
43: b←recur(tot-1,num)
44: if "null" is not in b then
45:     for j in a do
46:         res.append("0"+j)
47:     endfor
48: endif
49: return res
```

方案 2 算法设计的核心用伪代码的形式表示如下：

Algorithm 3: 方案 2

Input: 总比特位数 n，生成多项式 $P(x)$ 和期望比特翻转位数 i

Output: 能通过 CRC 校验的所有 i 位比特翻转的情况

```
calculate k and r
choice_bin ← recur(n,i)
choice_poly ← [change_binaryflow_to_polynomial(j) for j in choice_bin]
for E(x) in choice_poly do
    if P(x)|E(x) then
        E.append( E(x) )
    endif
end for
if isEmpty(E) then
    print('No possible case that can pass the CRC!')
else then
    print(str(len(E))+' cases that can pass the CRC:')
    output details of all the cases according to the global list E
endif
return None
```

可以看到此时的循环迭代次数为$\binom{n}{i}=\frac{n!}{i!(n-i)!}$。由 stirling 公式

$$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

可得其复杂度是$O(n^n)$的，比方案 1 的$O(2^n)$复杂度高。

2.5 实验结果

具体实现语言采用 SageMath 编程，代码详见附件。

运行例 1，期望比特翻转位数 i=3，结果如下：

```
input whole length of message (demo: 10)  n =
10
input the P polynomial, need to match with 1(0|1)* demo:11001.  P =
11001
Expecting how many bits to be inverted,i.e 0to1,1to0? input a number(demo:3):
3
---using plan1---
testing...
testing done.
9 cases that can pass the CRC:
case1: bit flipping 6 7 and 10th bits(i.e. E(x) = 0000011001)
case2: bit flipping 5 6 and 9th bits(i.e. E(x) = 0000110010)
case3: bit flipping 4 5 and 8th bits(i.e. E(x) = 0001100100)
case4: bit flipping 3 4 and 7th bits(i.e. E(x) = 0011001000)
case5: bit flipping 2 3 and 6th bits(i.e. E(x) = 0110010000)
case6: bit flipping 2 4 and 10th bits(i.e. E(x) = 0101000001)
case7: bit flipping 1 2 and 5th bits(i.e. E(x) = 1100100000)
case8: bit flipping 1 3 and 9th bits(i.e. E(x) = 1010000010)
case9: bit flipping 1 8 and 10th bits(i.e. E(x) = 1000000101)
Remark: counting from left to right,beginning from 1
```

图 1：例 1 运行结果（比特翻转位数为 3）

可以看到程序选择了方案 1 并输出了所有可能的通过 CRC 校验的情况，共 9

种。其中的 case1 即对应了例 1 中的 $E(x)$，即翻转了第 6、第 7 和第 10 位比特。

此时方案 1 需要迭代的次数为 $2^k-1=2^6-1=63$，方案 2 需要迭代的次数为 $\binom{n}{i}=\binom{10}{3}=120$，63<120，所以的确应选择方案 1 会更加快一些。程序运行结果符合预期。

如果例 1 中的生成多项式、n、k 和 r 参数均不变，改变 i=8，那么此时方案 1 需要迭代的次数仍为 63，而方案 2 需要迭代的次数变为 $\binom{10}{8}=45$，63>45，所以此时换成方案 2 反而会更加快一些。程序运行结果如下：

```
input whole length of message (demo: 10)  n =
10
input the P polynomial, need to match with 1(0|1)* demo:11001.  P =
11001
Expecting how many bits to be inverted,i.e 0to1,1to0? input a number(demo:3):
8
---using plan2---
testing...
testing done.
3 cases that can pass the CRC:
case1: bit flipping 1 2 3 5 6 7 8 and 9th bits(i.e. E(x) = 1110111110)
case2: bit flipping 1 2 4 5 7 8 9 and 10th bits(i.e. E(x) = 1101101111)
case3: bit flipping 2 3 4 6 7 8 9 and 10th bits(i.e. E(x) = 0111011111)
Remark: counting from left to right,beginning from 1
```

图 2：例 1 运行结果（比特翻转位数为 8）

三、总结与展望

本文首先明确定义了什么是发生了 CRC 漏检，然后理论分析何时会发生漏检，给出充要条件并给出证明。紧接着具体设计了指数级增长和组合数增长两种复杂度的算法方案。为了进行性能优化，用户给定 n、$P(x)$ 和比特翻转位数 i 后，程序先分别计算两种方案的迭代循环次数并决定采用次数少的方案。同时，对方案 2 的组合遍历也采取了一些方法优化了性能。

当然，这些算法均还有一定的性能优化进步空间，例如可以再找出一些肯定不会漏检或肯定会漏检的条件，并单独拎出进行特判等等。文献[1]中给出了几条这样的条件，未来会考虑在编程代码中加入这些条件判断，以期优化性能。

参 考 资 料

[1]李明军. CRC 的漏检率分析和铁路信号产品的安全性改进[C]//.列车运行控制系统技术交流会专刊.《铁道通信信号》编辑部（Editorial Department of Railway Signalling & Communication）,2013:97-100.

[2]程立辉,黄贻彬,付金华,徐洁.CRC 算法在计算机网络通信中的漏检分析[J].河南科学,2007(03):473-475.