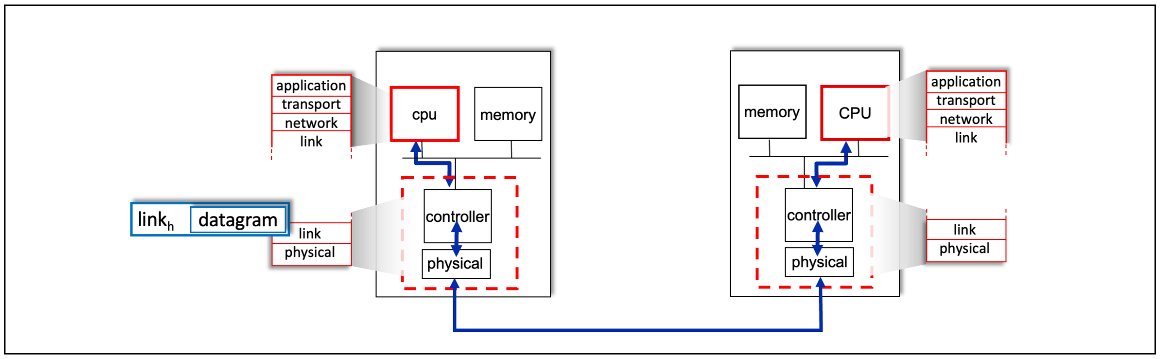
Class Notes

Chapter 6: The Link Layer and Local Area Networks (LANs)

High-level Introduction:

(1) Packets at the link layer are called “frames”. Recall our service model:



(2) Concerns for the layer-2:
   a. How to deliver data on adjacent links (wired or wireless or switched LAN etc.)
      i. Will need addressing at L2 layer. Example: Ethernet 48-bit address
      ii. Medium specific issues: For example: Half duplex or full duplex links?
   b. Reliability:
      i. Error detection and correction (Again?)
      ii. Why? We already did that at transport layer?
      iii. When to do it again at L2 and when not?
   c. Flow control
   d. Medium Access Control (MAC) for shared channels

(3) Physical and link layers are often implemented in hardware
   a. But link-layer interfaces with the software as well for sending / receiving data or to setup interrupts and DMAs (Direct Memory Access)

Error Detection:

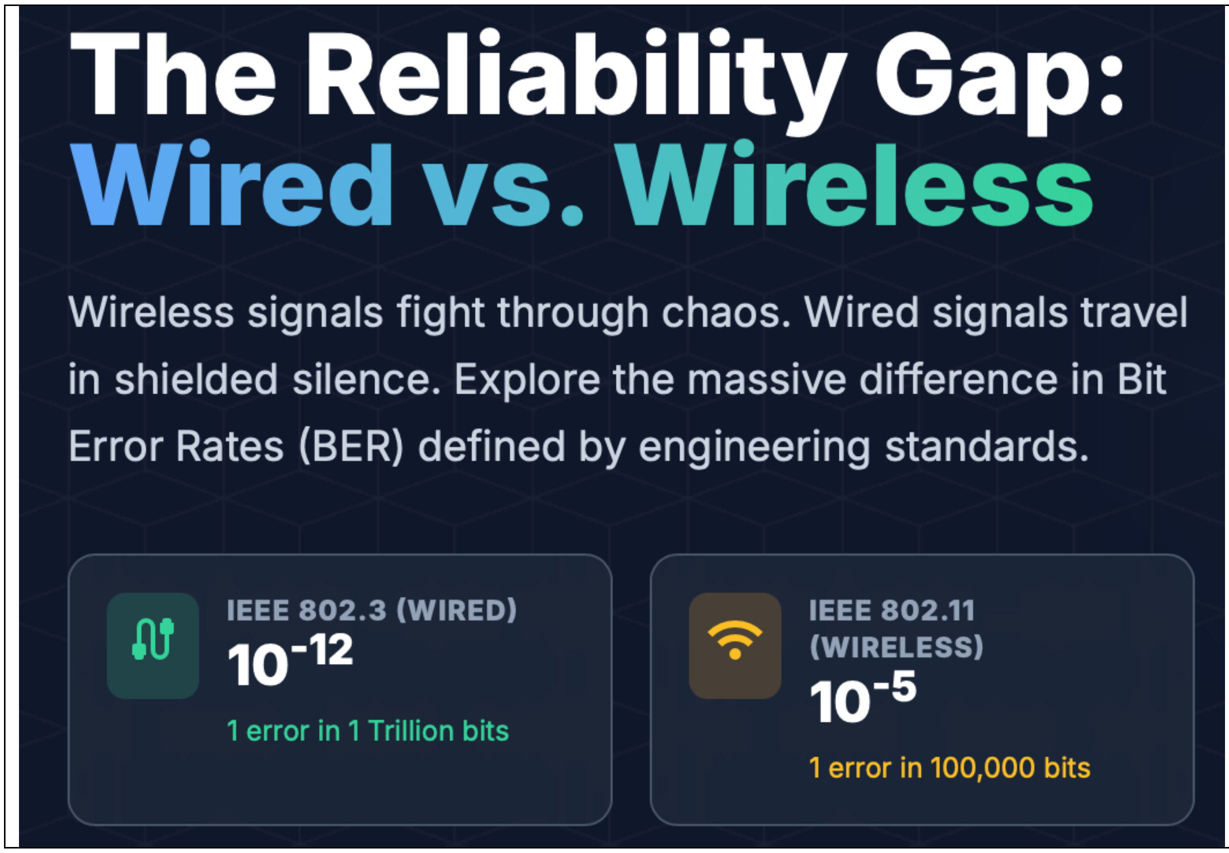
(1) At the physical layer we are concerned with transmitting and receiving bits.

(2) We tag on extra information for error detection or correction.
   a. Such extra data is overhead in terms of using link capacity, and also, we incur computation on sender and receiver sides to calculate them.

(3) We might not be able to detect all possible errors on the receiver side depending on the error detection scheme used. (Remind weaknesses of parity.).
   a. Tradeoff: Performance VS Low probability of evading bit errors
   b. Usually, we need more computation if we want higher reliability in terms of error detection

For Ethernet:
Min data size = 46 Bytes
Max data size = 1500 Bytes
CRC32 size = 4 Bytes

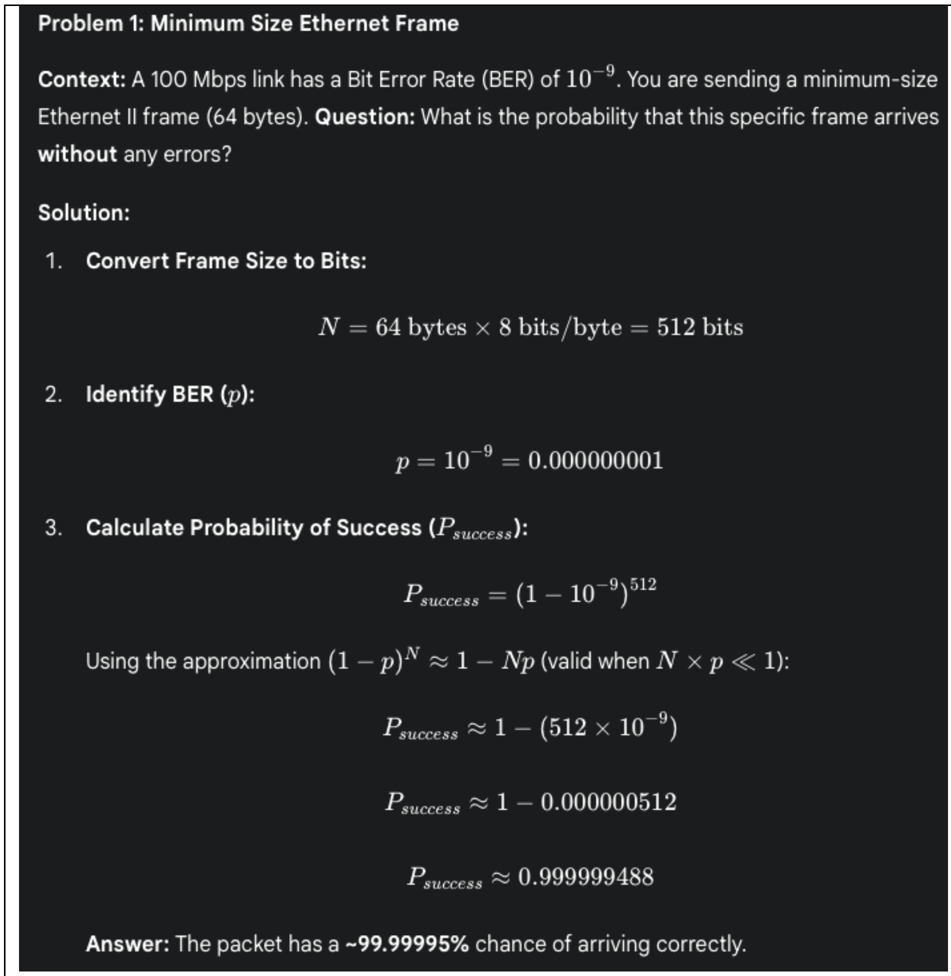
So 4 Bytes to detect errors in 1500 Bytes of data (~0.3% overhead due to CRC)


The Reliability Gap: Wired vs. Wireless

Wireless signals fight through chaos. Wired signals travel in shielded silence. Explore the massive difference in Bit Error Rates (BER) defined by engineering standards.

IEEE 802.3 (WIRED)
10^{-12}
1 error in 1 Trillion bits

IEEE 802.11 (WIRELESS)
10^{-5}
1 error in 100,000 bits



Problem 1: Minimum Size Ethernet Frame

Context: A 100 Mbps link has a Bit Error Rate (BER) of 10^{-9}. You are sending a minimum-size Ethernet II frame (64 bytes). **Question:** What is the probability that this specific frame arrives **without** any errors?

Solution:

1. **Convert Frame Size to Bits:**

$$N = 64 \text{ bytes} \times 8 \text{ bits/byte} = 512 \text{ bits}$$

2. **Identify BER (p):**

$$p = 10^{-9} = 0.000000001$$

3. **Calculate Probability of Success ($P_{success}$):**

$$P_{success} = (1 - 10^{-9})^{512}$$

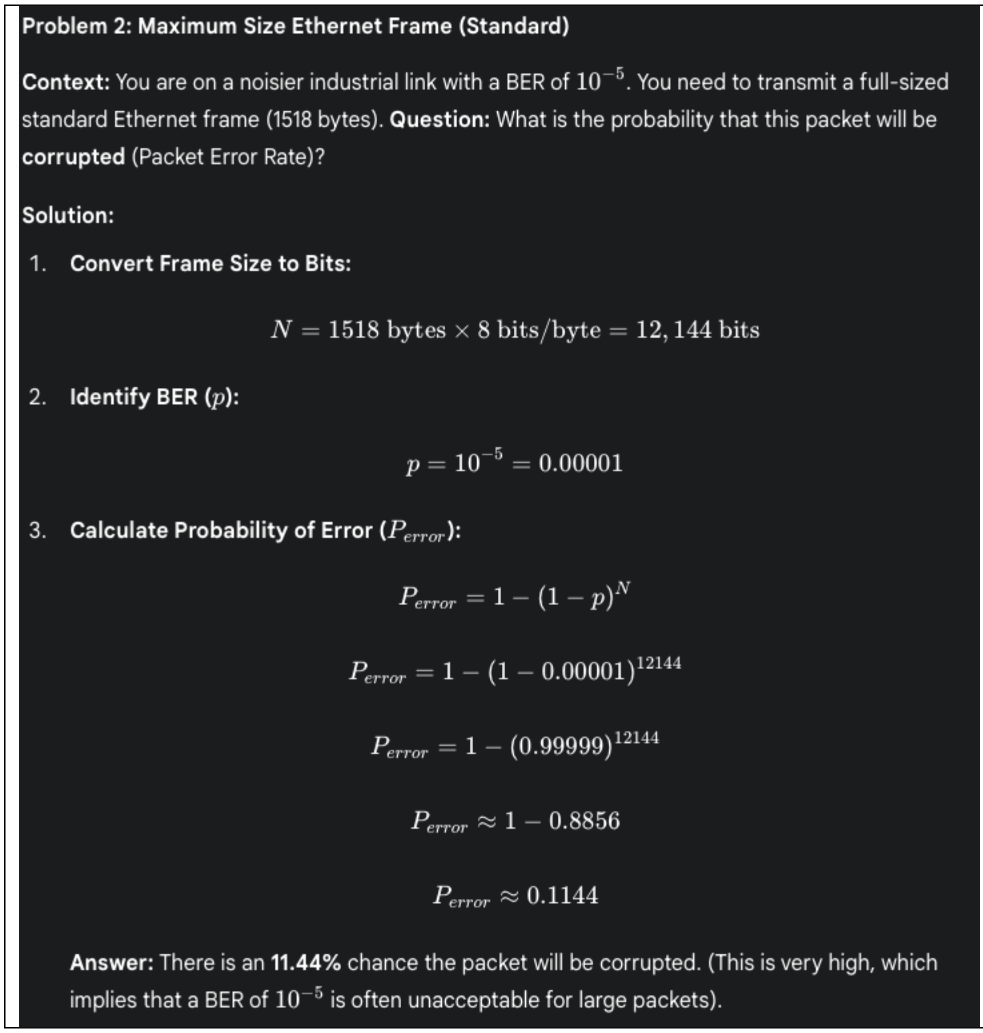
Using the approximation $(1-p)^N \approx 1 - Np$ (valid when $N \times p \ll 1$):

$$P_{success} \approx 1 - (512 \times 10^{-9})$$

$$P_{success} \approx 1 - 0.000000512$$

$$P_{success} \approx 0.999999488$$

Answer: The packet has a **~99.99995%** chance of arriving correctly.



Problem 2: Maximum Size Ethernet Frame (Standard)

Context: You are on a noisier industrial link with a BER of 10^{-5}. You need to transmit a full-sized standard Ethernet frame (1518 bytes). **Question:** What is the probability that this packet will be **corrupted** (Packet Error Rate)?

Solution:

1. **Convert Frame Size to Bits:**

$$N = 1518 \text{ bytes} \times 8 \text{ bits/byte} = 12,144 \text{ bits}$$

2. **Identify BER (p):**

$$p = 10^{-5} = 0.00001$$

3. **Calculate Probability of Error (P_{error}):**

$$P_{error} = 1 - (1 - p)^N$$

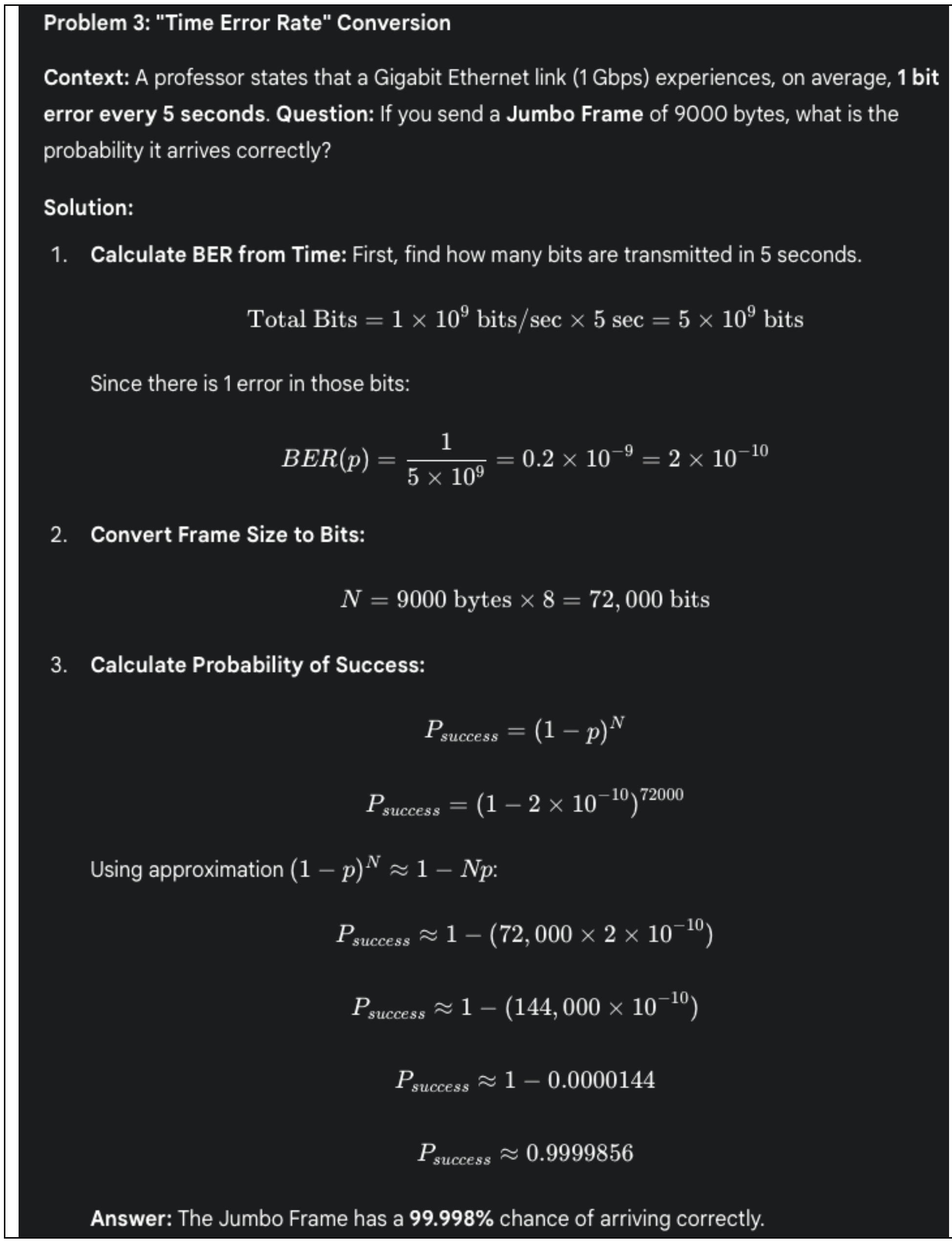
$$P_{error} = 1 - (1 - 0.00001)^{12144}$$

$$P_{error} = 1 - (0.99999)^{12144}$$

$$P_{error} \approx 1 - 0.8856$$

$$P_{error} \approx 0.1144$$

Answer: There is an **11.44%** chance the packet will be corrupted. (This is very high, which implies that a BER of 10^{-5} is often unacceptable for large packets).



Problem 3: "Time Error Rate" Conversion

Context: A professor states that a Gigabit Ethernet link (1 Gbps) experiences, on average, **1 bit error every 5 seconds**. **Question:** If you send a **Jumbo Frame** of 9000 bytes, what is the probability it arrives correctly?

Solution:

1. **Calculate BER from Time:** First, find how many bits are transmitted in 5 seconds.

$$\text{Total Bits} = 1 \times 10^9 \text{ bits/sec} \times 5 \text{ sec} = 5 \times 10^9 \text{ bits}$$

   Since there is 1 error in those bits:

$$BER(p) = \frac{1}{5 \times 10^9} = 0.2 \times 10^{-9} = 2 \times 10^{-10}$$

2. **Convert Frame Size to Bits:**

$$N = 9000 \text{ bytes} \times 8 = 72,000 \text{ bits}$$

3. **Calculate Probability of Success:**

$$P_{success} = (1-p)^N$$

$$P_{success} = (1 - 2 \times 10^{-10})^{72000}$$

   Using approximation $(1-p)^N \approx 1 - Np$:

$$P_{success} \approx 1 - (72,000 \times 2 \times 10^{-10})$$

$$P_{success} \approx 1 - (144,000 \times 10^{-10})$$

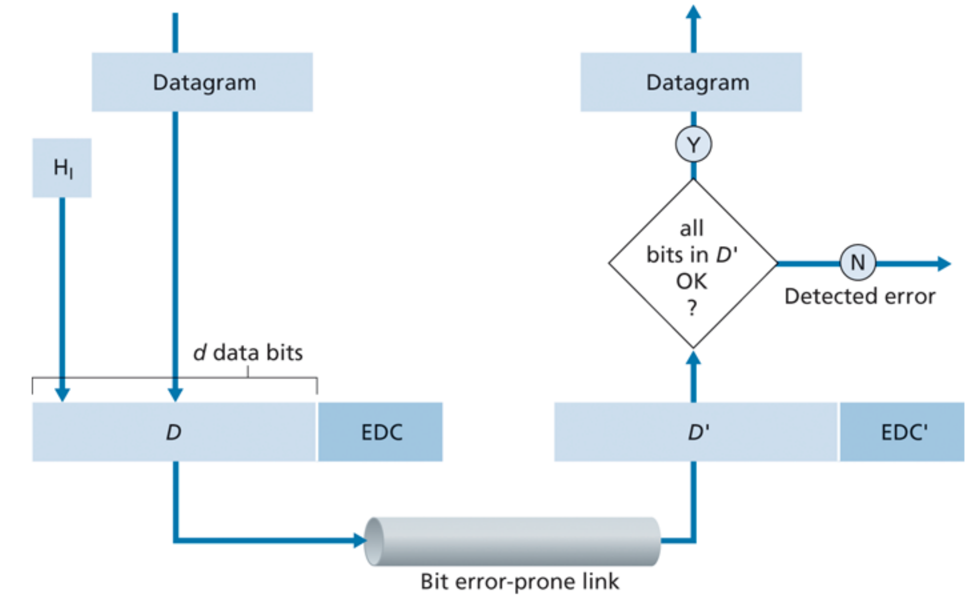
$$P_{success} \approx 1 - 0.0000144$$

$$P_{success} \approx 0.9999856$$

   Answer: The Jumbo Frame has a **99.998%** chance of arriving correctly.

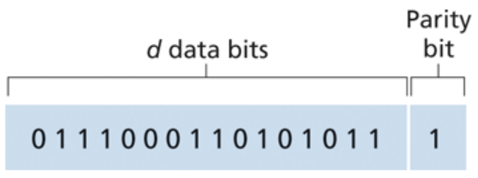
(4) A typical setup is as follows:

Figure 6.3 Error-detection and -correction scenario



(5) Parity Checks: Single bit used. Example even-parity:

Figure 6.4 One-bit even parity

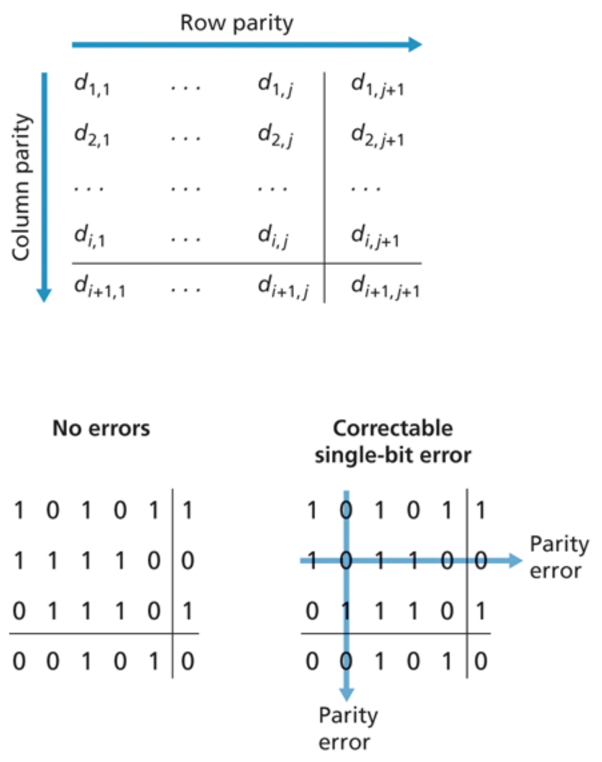


(6) Can't detect even number of bit flips.
  a. Burst errors (many adjacent bits flipping) is common in networking
  b. Aside: Be careful that many error detection and correction schemes are designed for random transmission errors and **not to guard against malicious attacks**

Error Correction:

(1) We can use two-dimensional parity to detect + correct 1-bit errors and to detect any 2-bit errors.

Figure 6.5 Two-dimensional even parity



Row parity
Column parity
$d_{1,1}$ . . . $d_{1,j}$ | $d_{1,j+1}$
$d_{2,1}$ . . . $d_{2,j}$ | $d_{2,j+1}$
. . . . | . . .
$d_{i,1}$ . . . $d_{i,j}$ | $d_{i,j+1}$
$d_{i+1,1}$ . . . $d_{i+1,j}$ | $d_{i+1,j+1}$
No errors
1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
0 0 1 0 1 | 0
Correctable single-bit error
1 0 1 0 1 | 1
1 0 1 1 0 | 0
0 1 1 1 0 | 1
0 0 1 0 1 | 0
Parity error
Parity error

(2) **Forward Error Correction (FEC):** The ability of the receiver to both detect and correct errors is known as forward error correction (FEC).

a. These techniques are commonly used in audio storage and playback devices such as audio CDs.

b. It is attractive that receiver could not only detect but correct errors because doing so we will not need to send a NAK and sender will not need to re-transmit the data.

c. Especially useful for long-propagation delay links (Example: NASA's MARS rover sending info to the Houston ground station.)

Internet Checksums:

(1) We encountered them:

a. In transport layer for UDP and TCP (16-bit field)

b. In IPv4 (for header only). IPv6 does not use it.

(2) Checksums are relatively easy to compute (they are sums!). But provide relatively weak protection against errors.

a. Example: If we swap two 16 bit words (which are being check summed), the internet checksum will still be the same because internet checksum uses one's complement which is both commutative and associative.

Cyclic Redundancy Checks (CRC) (AKA: Polynomial Codes)

(1) Commonly used at the link layer due to their ability to catch many errors.
(2) For Ethernet, CRC is calculated on (destination mac, source mac, type, payload).

High level idea:

On the receiver:

Receiver considers the full packet (packet header + payload + sender size CRC32) as a polynomial M(x).

Biggest Ethernet packet size = 1518 Bytes = 12144
So such polynomial will be of degree 12143

Ethernet gives us a divisor polynomial

- CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$

If the module-2 division result of these two polynomials (M(x) / CRC-32) is 0, that implies that data is correct.
(**Caution**: There are some probabilities involved with above statement and some errors can escape CRC-32. But probability of such happening is low.)

On the sender:
We need to figure out CRC32 values such that combined message (header of Ethernet + payload + crc32) is evenly divisibly by the generator polynomial.

Generator polynomial is not random:
We design the Generator Polynomial deliberately—it's not just random numbers. It is "engineered" to be incompatible with the mathematical structure of common errors (single bit, odd numbers, and bursts).

1. The Fundamental Concept: $P(x) + E(x)$

The text establishes a model for how errors occur in transmission using polynomial arithmetic (specifically Modulo-2 arithmetic).

- $P(x)$**:** The valid transmitted message (which includes the CRC). By definition, this is perfectly divisible by the generator $C(x)$.
- $E(x)$**:** The "Error Polynomial." This represents the bits that were flipped during transmission.
- $P(x) + E(x)$**:** This is what the receiver actually gets.

The Logic: When the receiver divides the incoming data by $C(x)$, it is effectively calculating:

$$\frac{P(x) + E(x)}{C(x)} = \frac{P(x)}{C(x)} + \frac{E(x)}{C(x)}$$

Since $\frac{P(x)}{C(x)}$ is always zero (valid messages have no remainder), the **only** thing that matters is $\frac{E(x)}{C(x)}$.

The Goal: We want to pick a $C(x)$ that **CANNOT** divide $E(x)$ evenly. If $E(x)$ is divisible by $C(x)$, the remainder is zero, and the receiver thinks the corrupted packet is valid (an undetected error).

2. The Four "Safety Rules" for Choosing $C(x)$

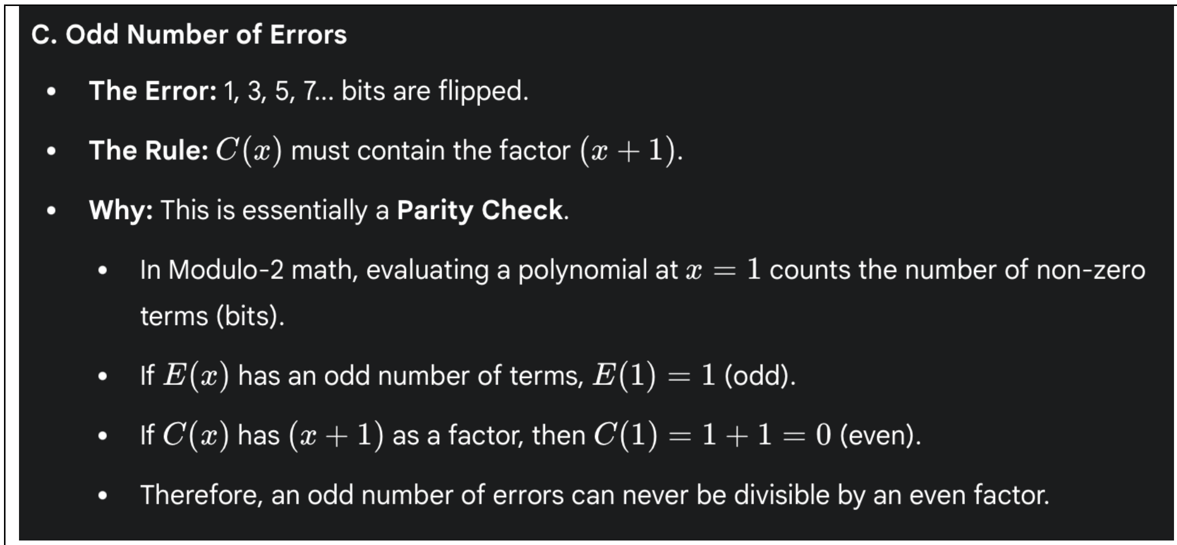
The image lists four types of errors and the mathematical property required in $C(x)$ to catch them.

A. Single-Bit Errors

- **The Error:** Only one bit flips at position i. Mathematically, $E(x) = x^i$.
- **The Rule:** If $C(x)$ has at least two terms (e.g., $x^k + 1$), it can never divide x^i evenly.
- **Why:** In polynomial division, a monomial (x^i) cannot be divided by a polynomial with multiple terms ($x^k + \cdots + 1$) without leaving a remainder.

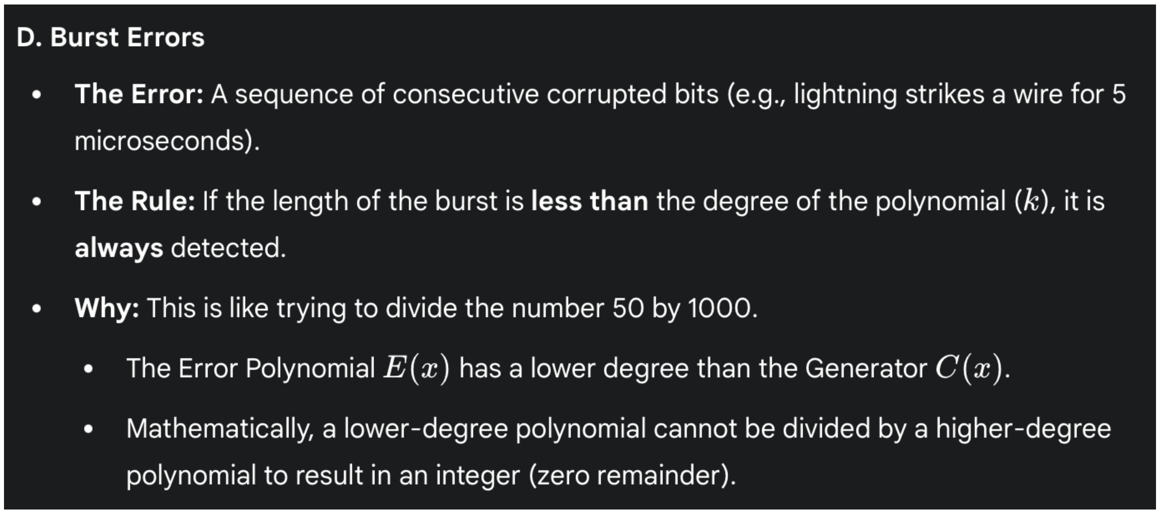
B. Double-Bit Errors

- **The Error:** Two bits flip at positions i and j. Mathematically, $E(x) = x^i + x^j$.
- **The Rule:** $C(x)$ is chosen specifically so it does not divide polynomials of this form up to a certain message length.
- **Note:** This is why standard polynomials (like CRC-32) are "Primitive Polynomials." They are mathematically proven not to factor into these simple two-term error shapes easily.



C. Odd Number of Errors

- **The Error:** 1, 3, 5, 7... bits are flipped.
- **The Rule:** $C(x)$ must contain the factor $(x+1)$.
- **Why:** This is essentially a **Parity Check**.
  - In Modulo-2 math, evaluating a polynomial at $x=1$ counts the number of non-zero terms (bits).
  - If $E(x)$ has an odd number of terms, $E(1)=1$ (odd).
  - If $C(x)$ has $(x+1)$ as a factor, then $C(1)=1+1=0$ (even).
  - Therefore, an odd number of errors can never be divisible by an even factor.

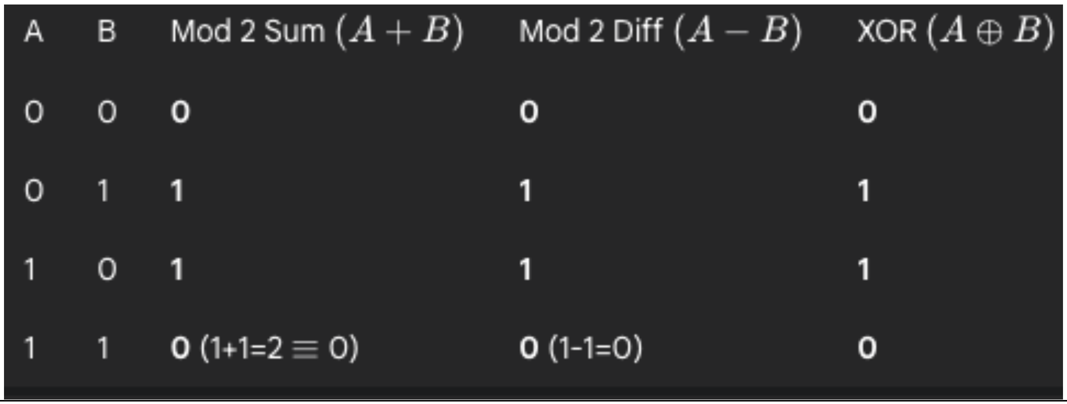


D. Burst Errors

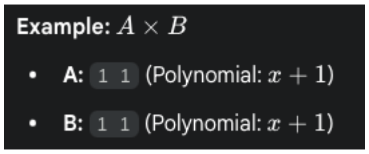
- **The Error:** A sequence of consecutive corrupted bits (e.g., lightning strikes a wire for 5 microseconds).
- **The Rule:** If the length of the burst is **less than** the degree of the polynomial (k), it is **always** detected.
- **Why:** This is like trying to divide the number 50 by 1000.
  - The Error Polynomial $E(x)$ has a lower degree than the Generator $C(x)$.
  - Mathematically, a lower-degree polynomial cannot be divided by a higher-degree polynomial to result in an integer (zero remainder).

Some math background for modulo-2 arithmetic:

Modulo-2 addition is same as subtraction and same as doing XOR

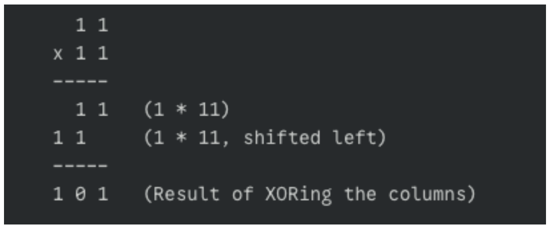


A	B	Mod 2 Sum $(A+B)$	Mod 2 Diff $(A-B)$	XOR $(A \oplus B)$
0	0	**0**	**0**	**0**
0	1	**1**	**1**	**1**
1	0	**1**	**1**	**1**
1	1	**0** (1+1=2 $\equiv$ 0)	**0** (1-1=0)	**0**

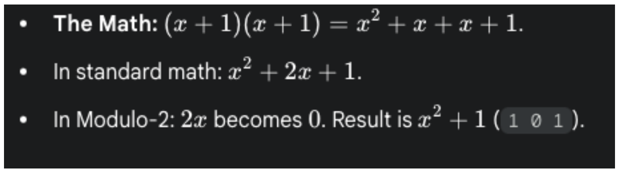


Example: $A \times B$

- **A:** `1 1` (Polynomial: $x + 1$)
- **B:** `1 1` (Polynomial: $x + 1$)



```
   1 1
 x 1 1
 -----
   1 1   (1 * 11)
 1 1     (1 * 11, shifted left)
 -----
 1 0 1   (Result of XORing the columns)
```

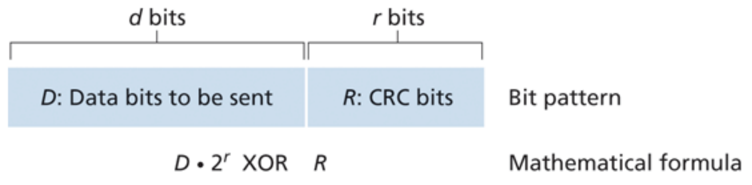


- **The Math:** $(x+1)(x+1) = x^2 + x + x + 1$.
- In standard math: $x^2 + 2x + 1$.
- In Modulo-2: $2x$ becomes 0. Result is $x^2 + 1$ (`1 0 1`).

(3) CRC Algo:
   a. D = "d" data bits
   b. Both parties agree on a Generator G, having (r+1) bits but most significant (left-most bit should be 1.)
   c. Sender needs to append R (= r bits) with D (= d bits) such that (d+r) bits is fully divisible by G (modulo 2)

(4) At receiver:
   a. If received (d+r) bits are fully divisible by G (modulo 2), there are "no errors" in the data

Figure 6.6 **CRC**



(5) All CRC calculations are done in modulo-2 arithmetic:
   a. Without carries in addition
   b. Without borrows in subtraction

(6) Above implies that additions and subtractions are identical, and can be done using XOR of the operands
   a. 1011 + 0101 = 1011 – 0101 under modulo 2 arithmetic

(7) Multiplication and division are the same as in base-2 arithmetic, except that any required addition or subtraction is done without carries or borrows.

(8) $D.2^r$ XOR R = (d+r) bits

(9) How sender finds R? We want to find an R such that there is an n such that:

$$D \cdot 2^r \text{ XOR } R = nG$$

(10) Move R on the other side using XOR

$$D \cdot 2^r = nG \text{ XOR } R$$

(11) D, G and r are known, so above equation tells us that if we div by G on both sides, then on the right we will have n as quotient and R is the remainder.

$$R = \text{remainder } \frac{D \cdot 2^r}{G}$$

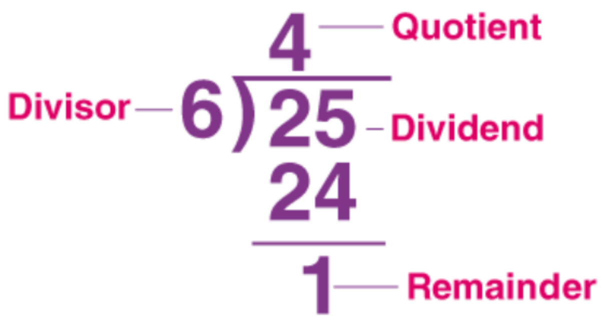
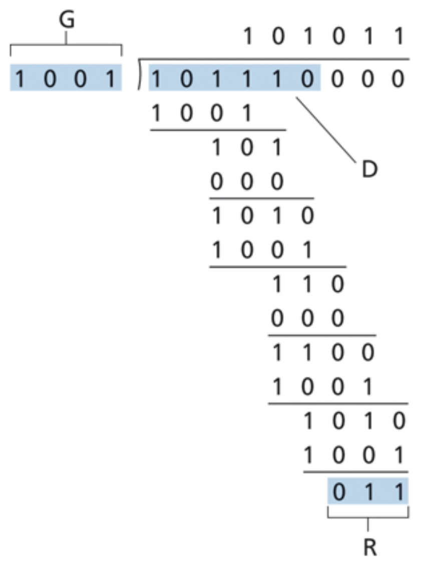


Image Source: Byjus: https://byjus.com/maths/quotient/

(12) Example: D = 101110 (6 bits), G = 1001, r = 3
(Just like in CRC32, the polynomial has 33 terms and the worst case remainder will be 32 bits.)

Figure 6.7 A sample CRC calculation



(13)CRC-32 (32 bit standard) adopted by many IEEE protocols uses the generator:

$$G_{\text{CRC-32}} = 100000100110000010001110110110111$$

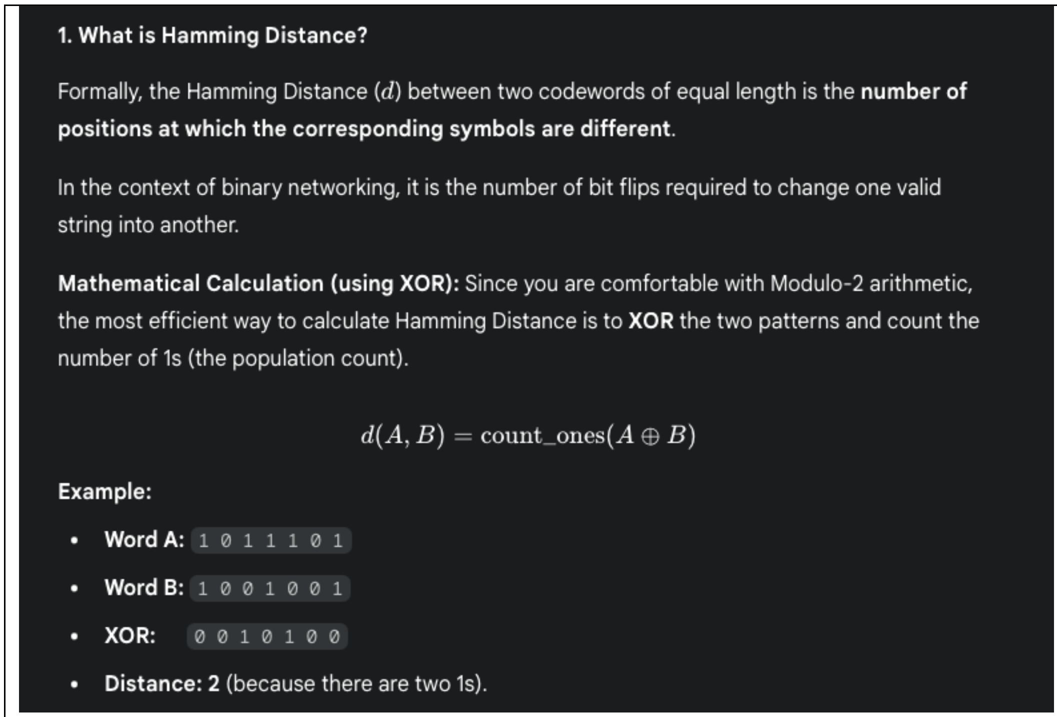
(14)CRC can detect burst errors of fewer than (r+1) bits

a. There are more details in the book:
b. CRC-32 can detect any number of odd bit errors.
c. A burst of errors of greater than (r+1) can be detected with probability $1\text{-}0.5^r$

(15)Error correction and detecting codes is a deep topic and a course in itself!

(16)Practice: What are the CRC bits R when D = 10011010, and r = 3 and G = 1001

(17)Hamming distance:



1. What is Hamming Distance?

Formally, the Hamming Distance (d) between two codewords of equal length is the **number of positions at which the corresponding symbols are different.**

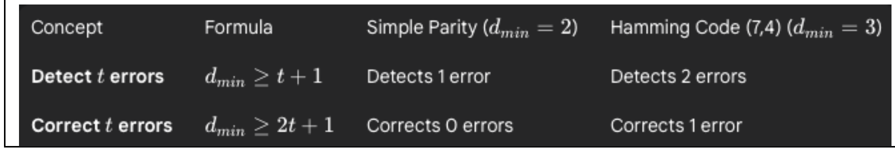
In the context of binary networking, it is the number of bit flips required to change one valid string into another.

Mathematical Calculation (using XOR): Since you are comfortable with Modulo-2 arithmetic, the most efficient way to calculate Hamming Distance is to **XOR** the two patterns and count the number of 1s (the population count).

$$d(A, B) = \text{count\_ones}(A \oplus B)$$

Example:

- **Word A:** `1 0 1 1 1 0 1`
- **Word B:** `1 0 0 1 0 0 1`
- **XOR:** `0 0 1 0 1 0 0`
- **Distance: 2** (because there are two 1s).



Concept	Formula	Simple Parity ($d_{min} = 2$)	Hamming Code (7,4) ($d_{min} = 3$)
Detect t errors	$d_{min} \geq t + 1$	Detects 1 error	Detects 2 errors
Correct t errors	$d_{min} \geq 2t + 1$	Corrects 0 errors	Corrects 1 error

For more details see:

(1) Computer networks: A systems approach, section 2.4
https://book.systemsapproach.org/direct/error.html

(2)