

Detecting and Correcting Bit Errors



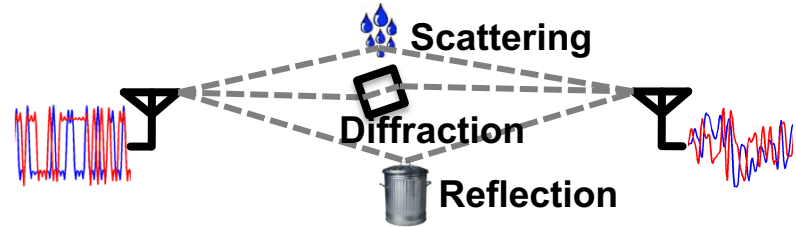
COS 463: Wireless Networks

Lecture 8

Kyle Jamieson

Bit errors on links

- Links in a network go through **hostile environments**
 - Both wired, and wireless:



- Consequently, **errors will occur on links**
 - Today: How can we detect and correct these errors?**
- There is **limited capacity** available on any link
 - Tradeoff** between **link utilization** & amount of **error control**

Today

1. Error control codes

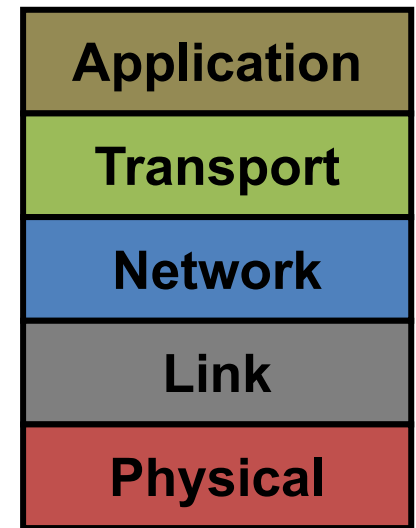
- Encoding and decoding fundamentals
- Measuring a code's error correcting power
- Measuring a code's overhead
- Practical error control codes
 - Parity check, Hamming block code

2. Error detection codes

- Cyclic redundancy check (CRC)

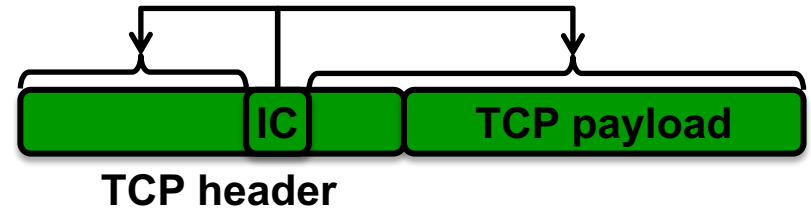
Where is error control coding used?

- The techniques we'll discuss today are **pervasive** throughout the internetworking stack
- Based on theory, but **broadly applicable** in practice, in other areas:
 - Hard disk drives
 - Optical media (CD, DVD, & c.)
 - Satellite, mobile communications
- In 463, we cover the “tip of the iceberg” in the Internetworking stack



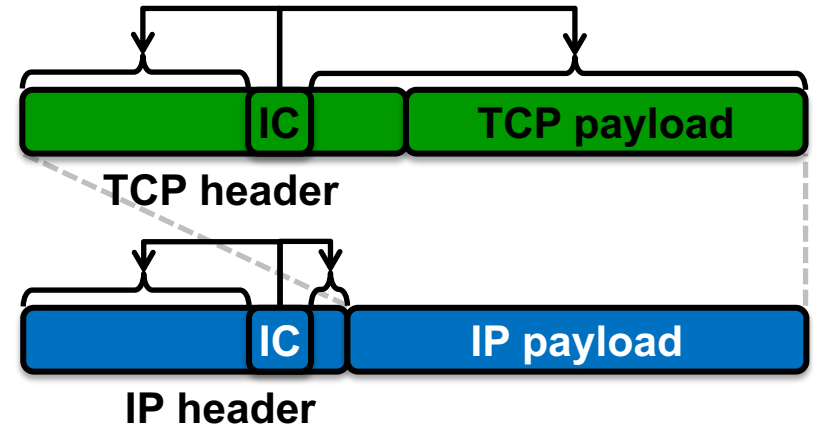
Error control in the Internet stack

- Transport layer
 - **Internet Checksum (IC)**
over TCP/UDP header, data



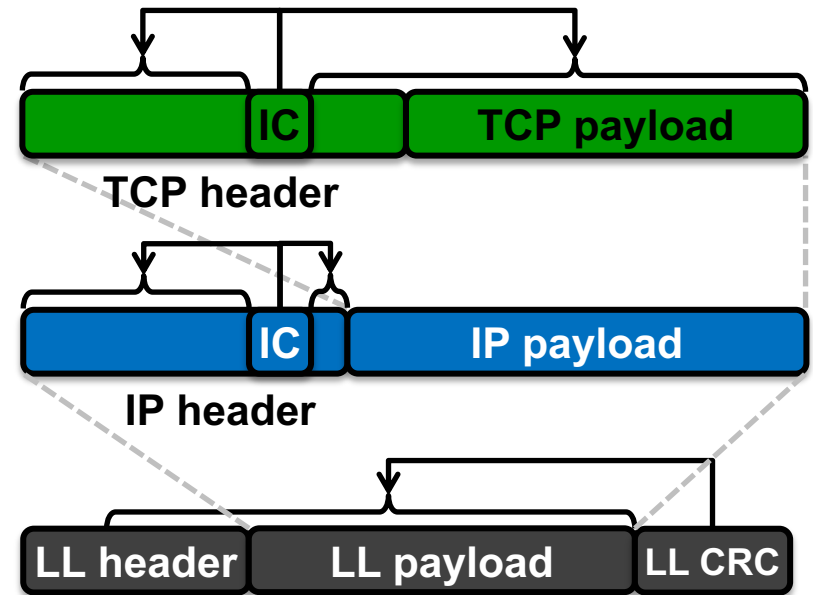
Error control in the Internet stack

- **Transport layer**
 - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
 - **IC** over IP header only



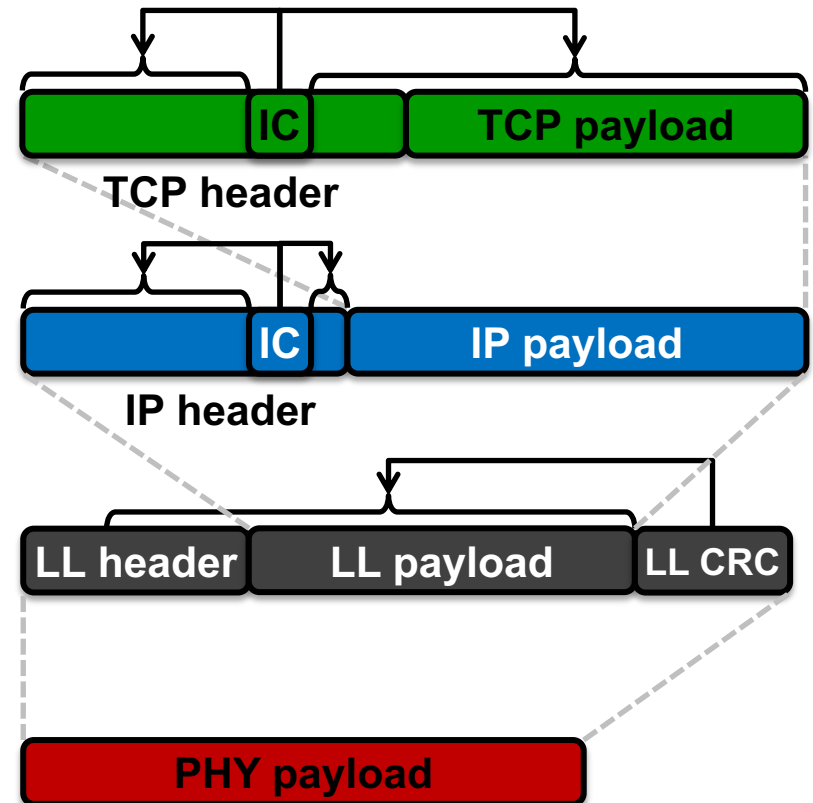
Error control in the Internet stack

- **Transport layer**
 - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
 - **IC** over IP header only
- **Link layer (L2)**
 - **Cyclic Redundancy Check (CRC)**



Error control in the Internet stack

- **Transport layer**
 - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
 - **IC** over IP header only
- **Link layer (L2)**
 - **Cyclic Redundancy Check (CRC)**
- **Physical layer (PHY)**
 - **Error Control Coding (ECC)**, or
 - **Forward Error Correction (FEC)**



Today

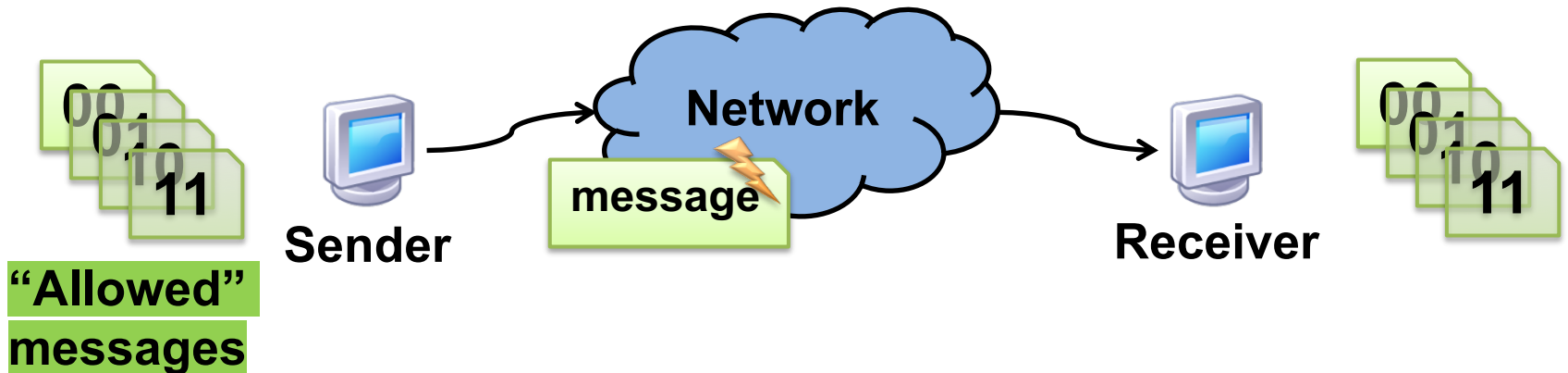
1. Error control codes

- **Encoding and decoding fundamentals**
- Measuring a code's error correcting power
- Measuring a code's overhead
- Practical error control codes
 - Parity check, Hamming block code

2. Error detection codes

- Cyclic redundancy check (CRC)

Error control: Motivation



- Every string of bits is an “allowed” message
 - Hence any **changes to the bits** (*bit errors*) the sender transmits produce “allowed” messages
- **Therefore without error control, receiver wouldn’t know errors happened!**

Error control: Key Ideas

- **Reduce the set of “allowed” messages**
 - **Not every string of bits** is an “allowed” message
 - Receipt of a **disallowed** string of bits means that the **message was garbled** in transit over the network
- We call an allowable message (of n bits) a **codeword**
 - **Not all** n -bit strings are codewords!
 - The remaining n -bit strings are **“space” between codewords**
- **Plan:** Receiver will **use that space** to both **detect** and **correct** errors in transmitted messages

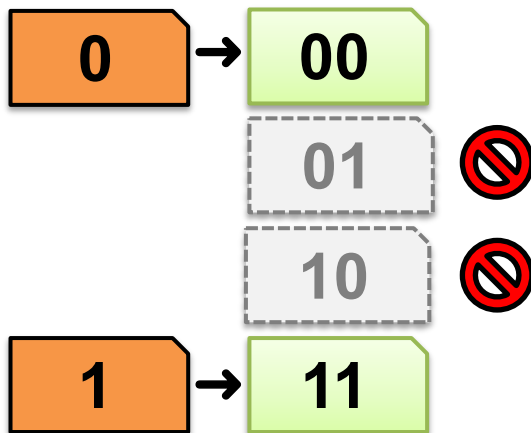
Encoding and decoding

- **Problem:** **Not every** string of bits is “allowed”
 - But we want to be **able to send any** message!
 - *How can we send a “disallowed” message?*
- **Answer: Codes,** as a sender-receiver **protocol**
 - The sender must **encode** its messages → codewords
 - The receiver then **decodes** received bits → messages
- The **relationship between messages and codewords** isn't always obvious!

A simple error-detecting code

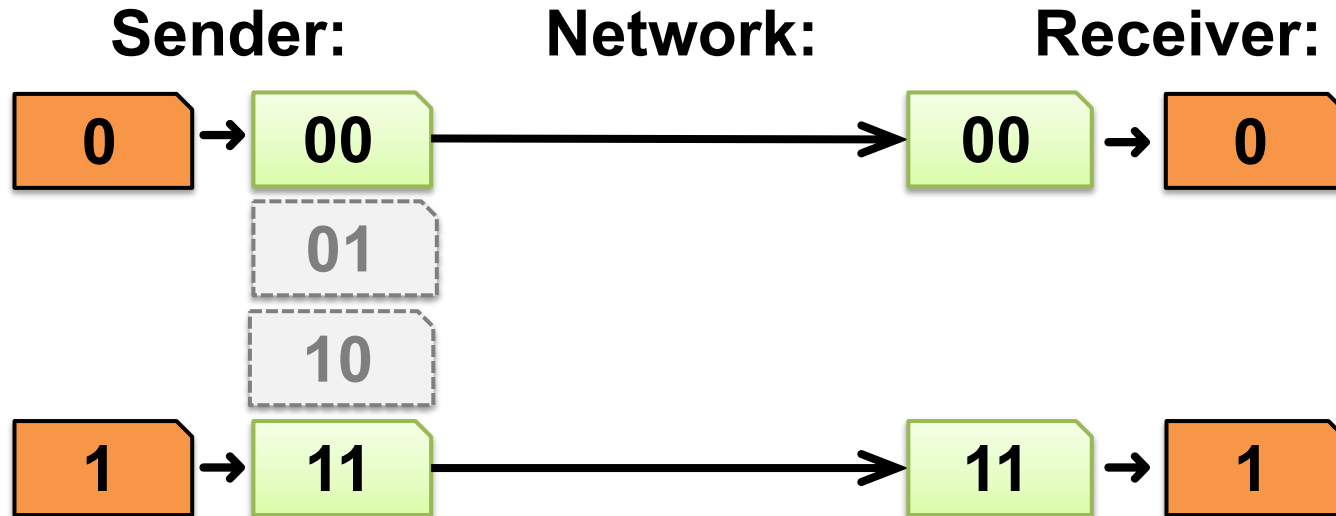
- Let's start simple: suppose messages are one bit long
- Take the message bit, and **repeat** it once
 - This is called a ***two-repetition code***

Sender:



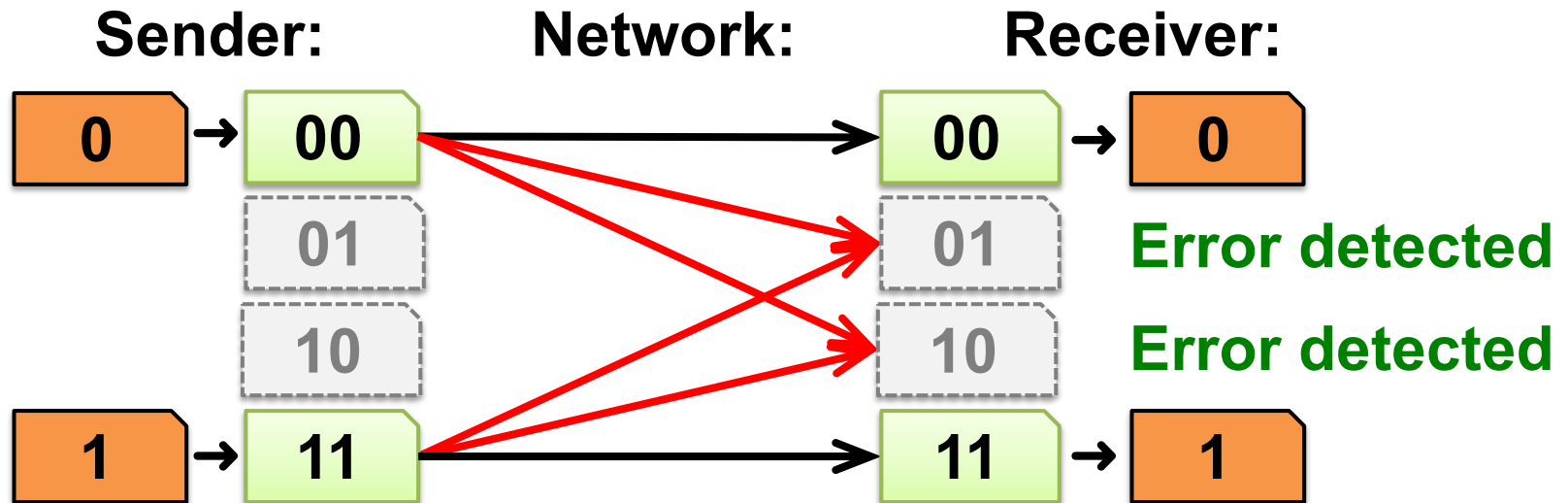
Receiving the two-repetition code

- Suppose the network causes **no bit error**
- Receiver **removes repetition** to **correctly decode** the message bits



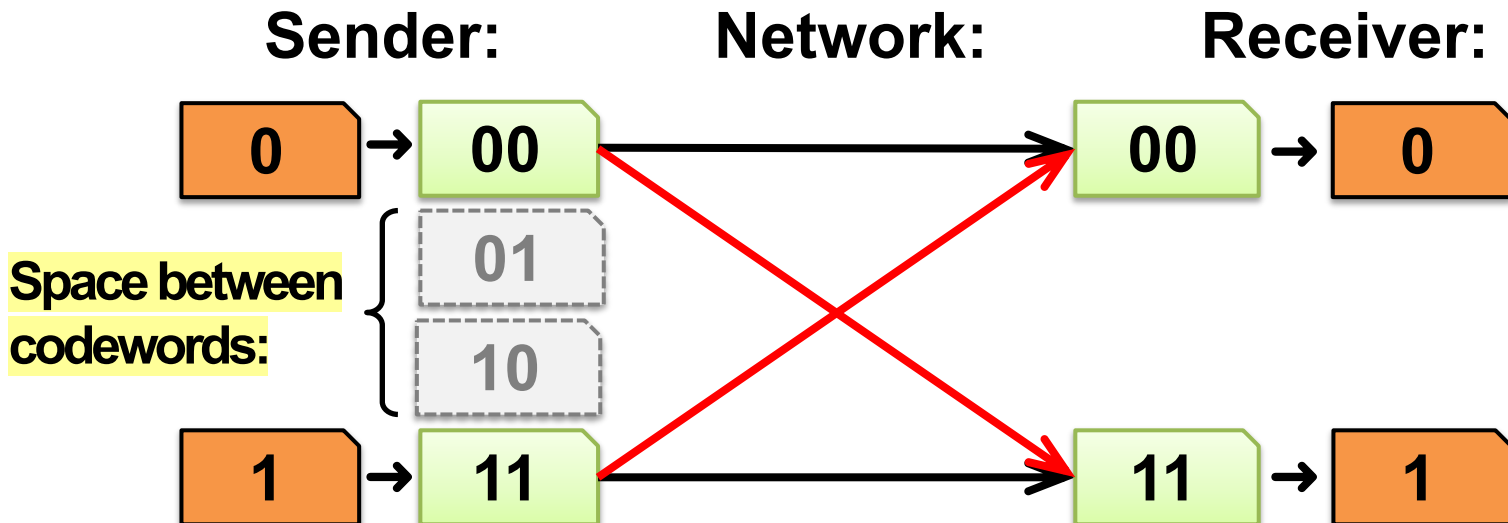
Detecting one bit error

- Suppose the network causes up to **one bit error**
- The receiver **can detect** the error:
 - It received a **non-codeword**
- Can the receiver **correct** the error?
 - **No!** The **other** codeword could have been sent as well



Reception with two bit errors

- Can receiver **detect** presence of **two bit errors**?
 - No**: It has no way of telling which codeword was sent!
 - Enough bit errors that the sent codeword “**jumped over**” the **space between** codewords



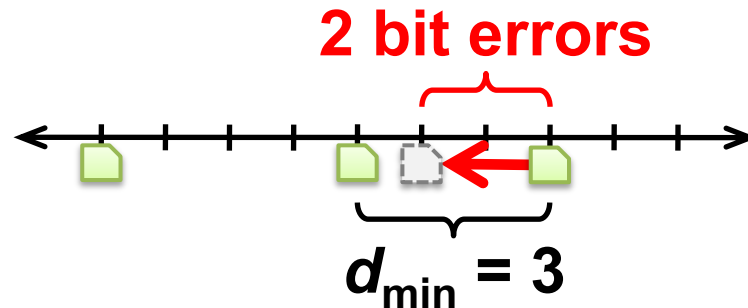
Hamming distance

- Measures the **number of bit flips** to change **one codeword into another**
- **Hamming distance** between two messages m_1, m_2 : The number of bit flips needed to change m_1 into m_2
- **Example:** Two bit flips needed to change codeword 00 to codeword 11, so they are Hamming distance of **two** apart:



How many bit errors can we detect?

- Suppose the **minimum Hamming distance** between any pair of codewords is d_{\min}
- Then, we **can detect** at most $d_{\min} - 1$ bit errors
 - Will land in space between codewords, as we just saw



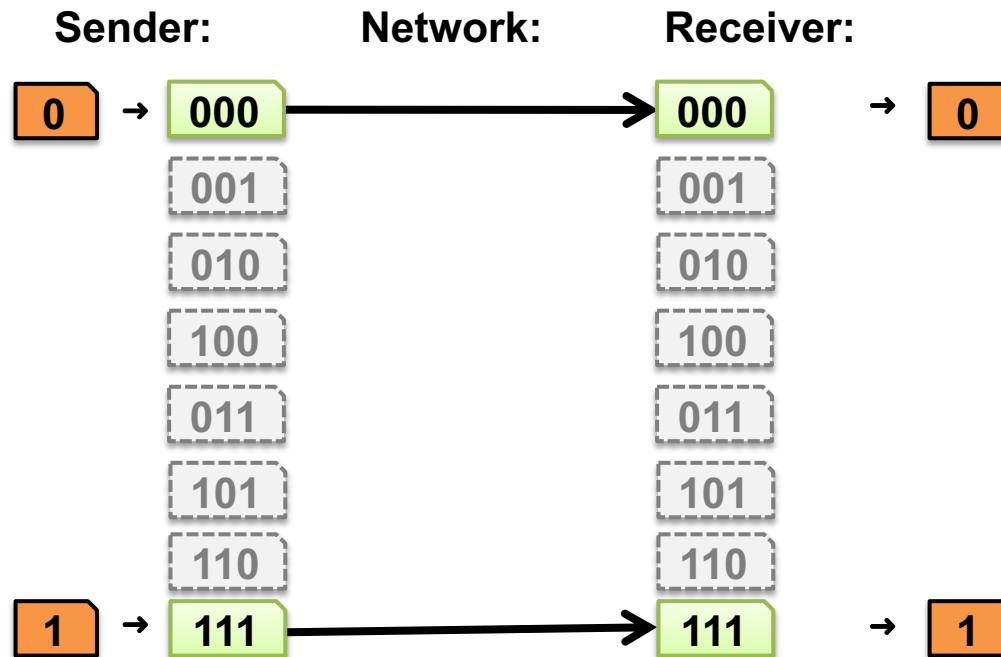
- Receiver will flag message as **“Error detected”**

Decoding error detecting codes

- **The receiver decodes** in a **two-step** process:
 1. Map **received bits → codeword**
 - **Decoding rule:** Consider all codewords
 - **Choose** the one that **exactly matches** the received bits
 - **Return “error detected”** if none match
 2. Map **codeword → source bits** and “error detected”
 - Use the **reverse map** of the sender

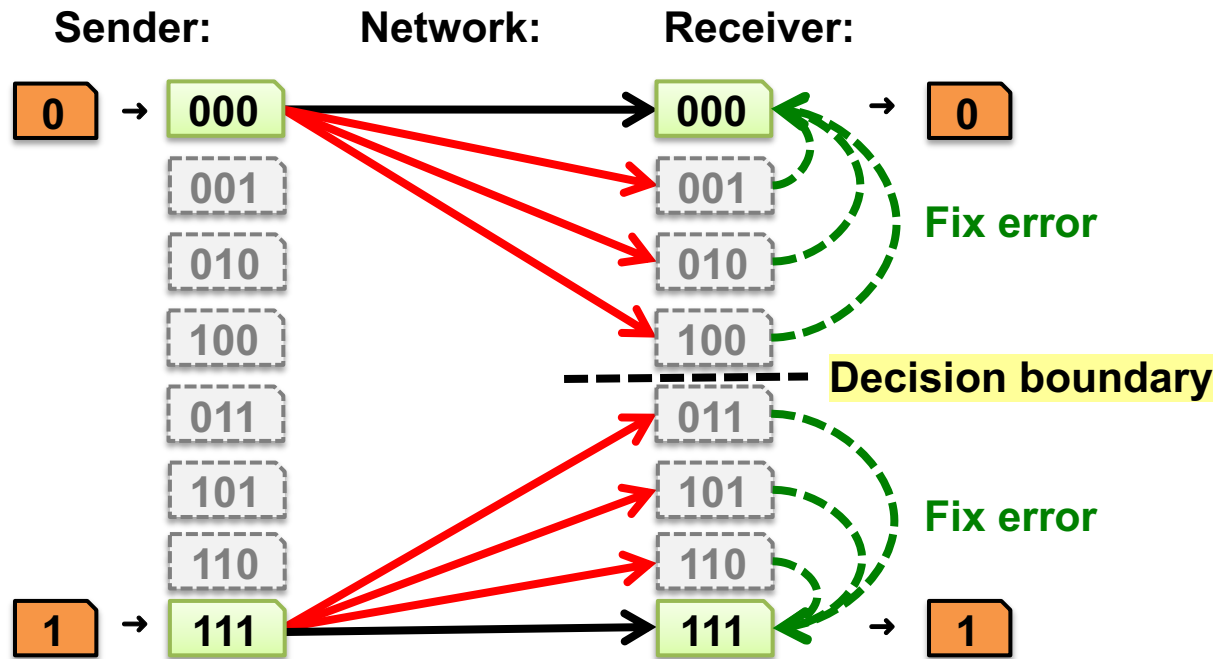
A simple error-correcting code

- Let's look at a **three-repetition** code
- If **no errors**, it works like the two-repetition code:



Correcting one bit error

- Receiver **chooses the closest codeword** (measured by Hamming distance) to the received bits
 - A **decision boundary exists halfway** between codewords

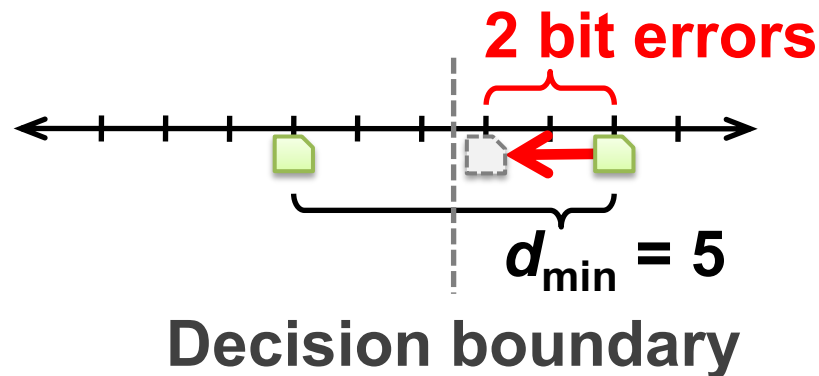


Decoding error correcting codes

- **The receiver decodes** in a two-step process:
 1. Map **received bits → codeword**
 - **Decoding rule:** Consider all codewords
 - **Choose one** with the **minimum Hamming distance** to the received bits
 2. Map **codeword → source bits**
 - Use the **reverse map** of the sender

How many bit errors can we correct?

- Suppose there is **at least d_{\min}** Hamming distance between any two codewords
- Then, we can **correct at most** $\left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$ **bit flips**
 - This many bit flips can't move received bits closer to another codeword, **across** the decision boundary:



Code rate

- Suppose **codewords** of length n , **messages** length k ($k < n$)
- The **code rate** $R = k/n$ is a fraction between 0 and 1
- So, we have a **tradeoff**:
 - **High-rate codes** (R approaching one) **correct fewer errors**, but **add less overhead**
 - **Low-rate codes** (R close to zero) **correct more errors**, but **add more overhead**

Today

1. Error control codes

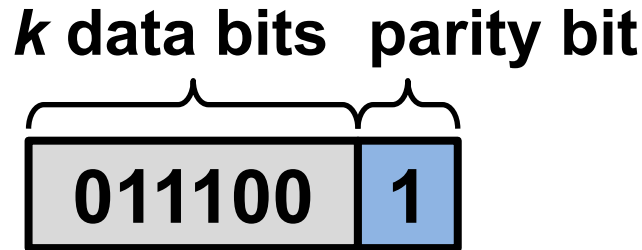
- Encoding and decoding fundamentals
- Measuring a code's error correcting power
- Measuring a code's overhead
- **Practical error control codes**
 - **Parity check, Hamming block code**

2. Error detection codes

- Cyclic redundancy check (CRC)

Parity bit

- Given a message of k data bits D_1, D_2, \dots, D_k , append a **parity bit P** to make a codeword of length $n = k + 1$
 - P is the exclusive-or of the data bits:
 - $P = D_1 \oplus D_2 \oplus \dots \oplus D_k$
 - Pick the parity bit so that **total number of 1's is even**



Checking the parity bit

- **Receiver:** counts number of 1s in received message
 - **Even:** received message is a codeword
 - **Odd:** isn't a codeword, and **error detected**
 - But receiver doesn't know where, so **can't correct**
- What about d_{\min} ?
 - Change one data bit \rightarrow change parity bit, so $d_{\min} = 2$
 - So parity bit **detects 1 bit error, corrects 0**
- Can we **detect and correct more errors**, in general?

Two-dimensional parity

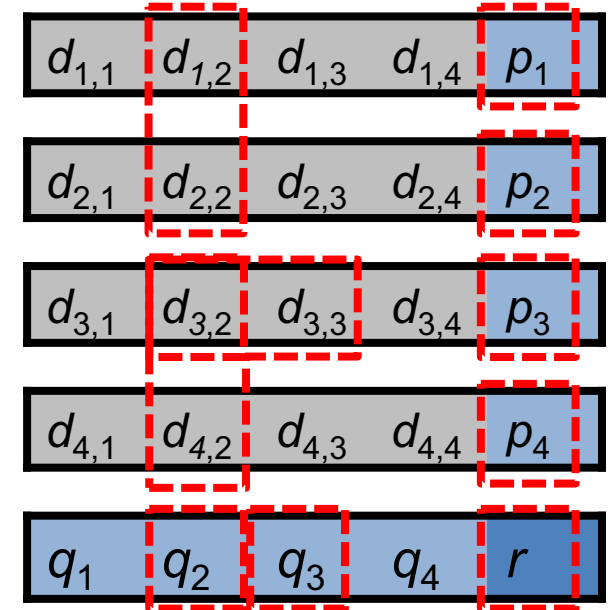
- Break up data into multiple rows
 - Start with normal parity **within** each row (p_i)
 - Do the same **down columns** (q_i)
 - Add a parity bit r covering row parities
- This example has rate 16/25

$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	$d_{1,4}$	p_1
$d_{2,1}$	$d_{2,2}$	$d_{2,3}$	$d_{2,4}$	p_2
$d_{3,1}$	$d_{3,2}$	$d_{3,3}$	$d_{3,4}$	p_3
$d_{4,1}$	$d_{4,2}$	$d_{4,3}$	$d_{4,4}$	p_4
q_1	q_2	q_3	q_4	r

$$\begin{aligned} p_j &= d_{j,1} \oplus d_{j,2} \oplus d_{j,3} \oplus d_{j,4} \\ q_j &= d_{1,j} \oplus d_{2,j} \oplus d_{3,j} \oplus d_{4,j} \\ r &= p_1 \oplus p_2 \oplus p_3 \oplus p_4 \end{aligned}$$

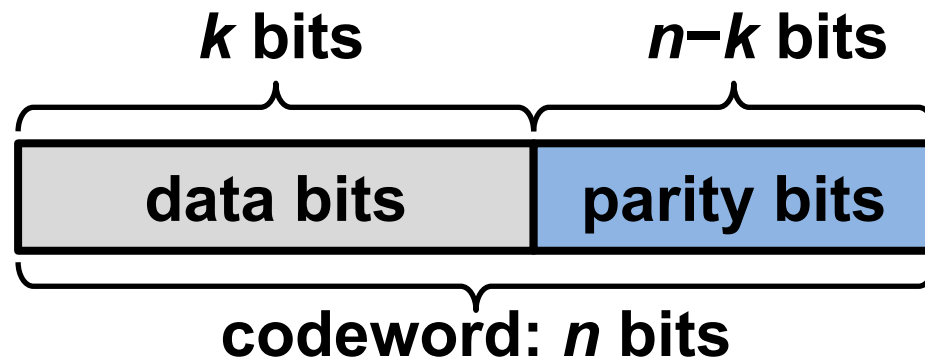
Two-dimensional parity: Properties

- Flip 1 data bit, **3 parity bits** flip
- Flip 2 data bits, **≥ 2 parity bits** flip
- Flip 3 data bits, **≥ 3 parity bits** flip
- Therefore, $d_{\min} = 4$, so
 - Can detect ≤ 3 bit errors
 - Can correct single-bit errors (*how?*)
- 2-D parity detects **most** four-bit errors



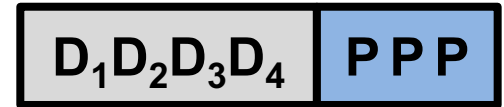
Block codes

- Let's **fully generalize the parity bit** for even more error detecting/correcting power
- Split message into **k -bit blocks**, and **add $n-k$ parity bits** to the end of each block:
 - This is called an **(n, k) block code**



A higher rate error correcting code?

- What if we **repeat the parity bit 3 ×** ?



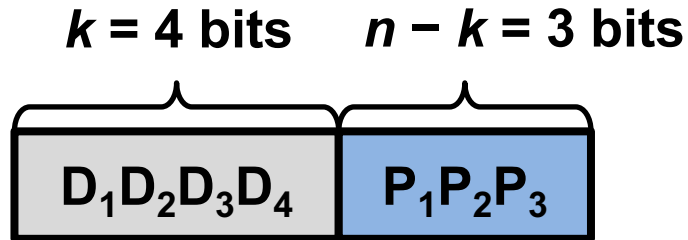
- $P = D_1 \oplus D_2 \oplus D_3 \oplus D_4$; $R = 4/7$

- Flip one data bit, all parity bits flip. So $d_{\min} = 4$?

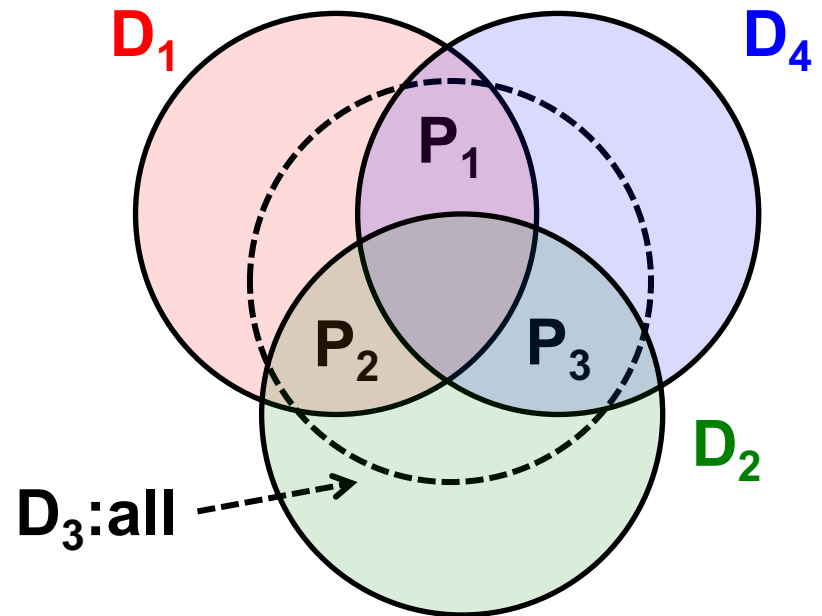
- **No!** Flip another data bit, all parity bits flip back to original values! So $d_{\min} = 2$

- **What happened?** Parity checks either **all failed or all succeeded**, giving **no additional information**

Hamming (7, 4) code

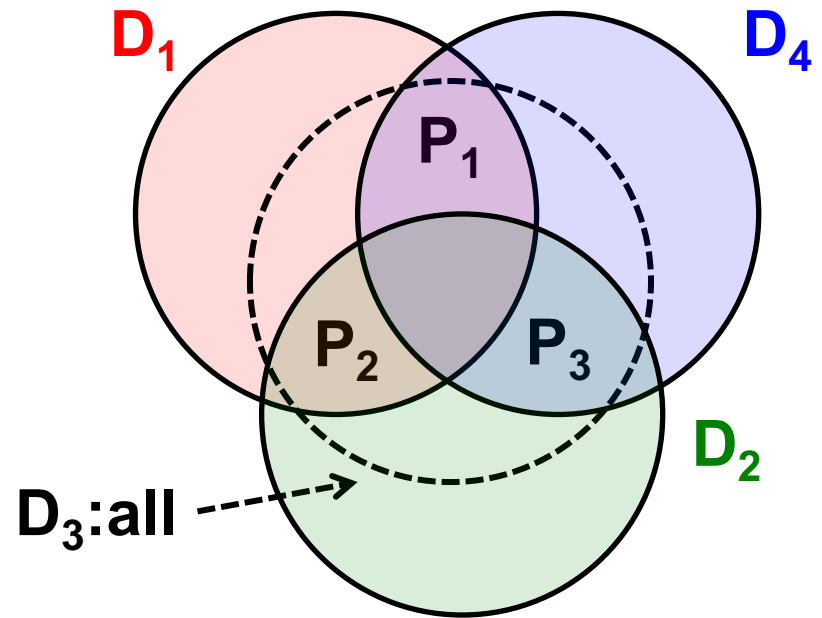


$$\begin{aligned} P_1 &= D_1 \oplus D_3 \oplus D_4 \\ P_2 &= D_1 \oplus D_2 \oplus D_3 \\ P_3 &= D_2 \oplus D_3 \oplus D_4 \end{aligned}$$



Hamming (7, 4) code: d_{\min}

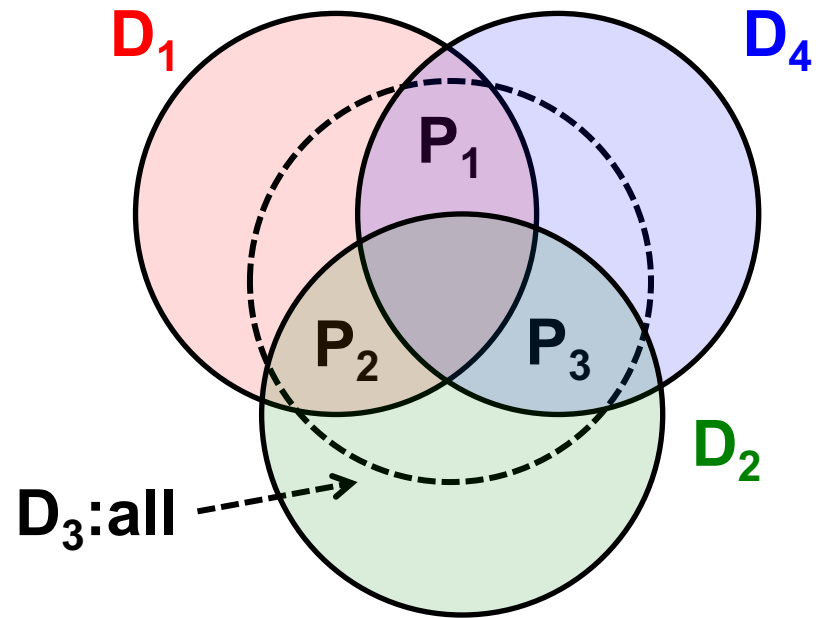
- **Change one data bit**, either:
 - ➔ Two P_i change, or
 - Three P_i change
- **Change two data bits**, either:
 - ➔ Two P_i change, or
 - One P_i changes



$d_{\min} = 3$: Detect 2 bit errors, correct 1 bit error

Hamming (7, 4): Correcting One Bit Error

- **Infer which** corrupt bit from which parity checks fail:
- P_1 and P_2 fail \Rightarrow Error in D_1
- P_2 and P_3 fail \Rightarrow Error in D_2
- $P_1, P_2,$ & P_3 fail \Rightarrow Error in D_3
- P_1 and P_3 fail \Rightarrow Error in D_4
- What if just **one** parity check fails?



Summary: Higher rate ($R = 4/7$) code **correcting one bit error**

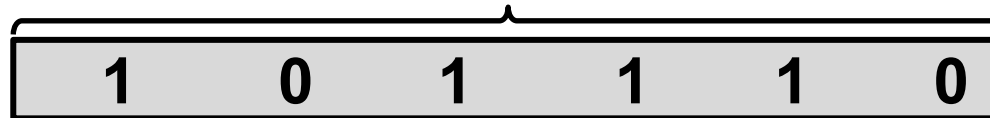
Today

1. Error control codes
- 2. Error detection codes**
 - Cyclic redundancy check (CRC)**

Cyclic redundancy check (CRC)

- Most popular method **error detecting code** at L2
 - Found in Ethernet, Wi-Fi, token ring, many many others
- Often implemented in hardware at the link layer
- Represent **k -bit messages** as **degree $k - 1$ polynomials**
 - **Each coefficient in the polynomial is either zero or one, e.g.:**

$k = 6$ bits of message



$$M(x) = 1x^5 + 0x^4 + 1x^3 + 1x^2 + 1x + 0$$

Modulo-2 Arithmetic

- Addition and subtraction are both **exclusive-or without carry or borrow**

Multiplication example:

$$\begin{array}{r} 1101 \\ 110 \\ \hline 0000 \\ 11010 \\ 110100 \\ \hline 101110 \end{array}$$

Division example:

$$\begin{array}{r} 1101 \\ 110 \overline{) 101110} \\ \underline{110} \downarrow \\ 111 \downarrow \\ \underline{110} \downarrow \\ 011 \downarrow \\ \underline{000} \downarrow \\ 110 \\ \underline{110} \\ 000 \end{array}$$

CRC at the sender

- $M(x)$ is our **message** of length k
 - **e.g.:** $M(x) = x^5 + x^3 + x^2 + x$ ($k = 6$)

1	0	1	1	1	0
---	---	---	---	---	---
 - Sender and receiver agree on a **generator** polynomial $G(x)$ of degree $g - 1$ (**i.e., g bits**)
 - **e.g.:** $G(x) = x^3 + 1$ ($g = 4$)

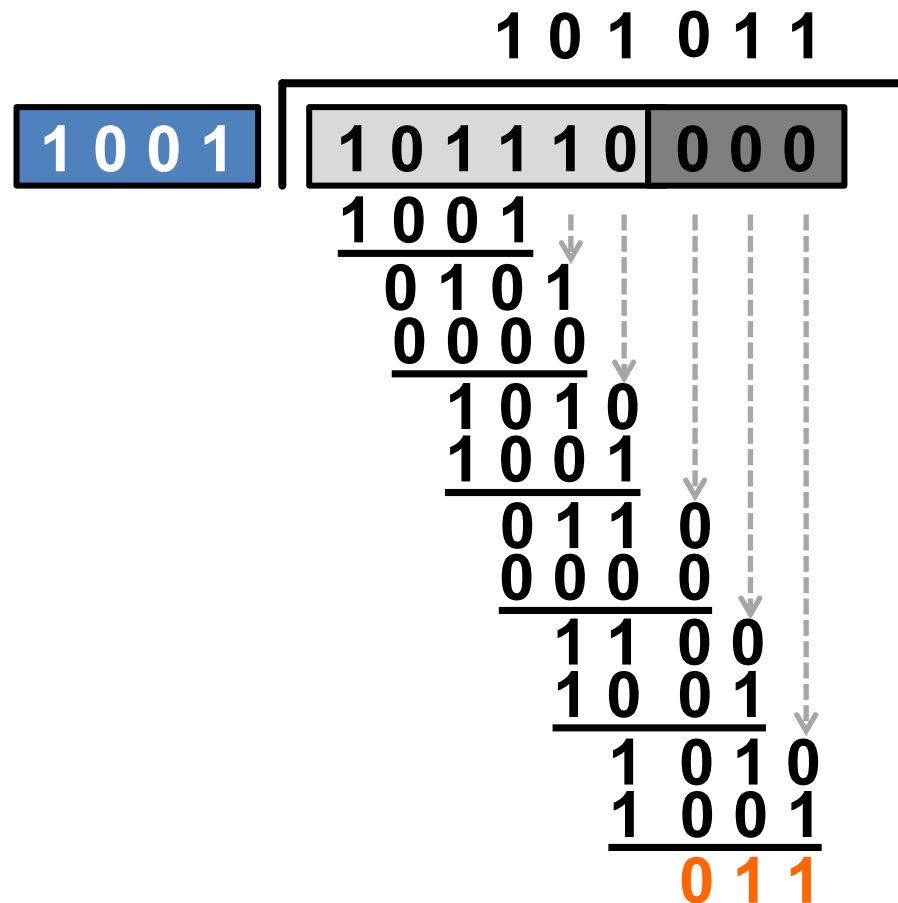
1	0	0	1
---	---	---	---
1. Calculate **padded message** $T(x) = M(x) \cdot x^{g-1}$
 - **i.e.,** right-pad with $g - 1$ zeroes
 - **e.g.:** $T(x) = M(x) \cdot x^3 = x^8 + x^6 + x^5 + x^4$

1	0	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---

CRC at the sender

2. Divide padded message $T(x)$ by generator $G(x)$

- The remainder $R(x)$ is the CRC:



$$R(x) = x + 1$$

CRC at the sender

3. The sender transmits codeword $C(x) = T(x) + R(x)$
- *i.e.*, the sender transmits the original message with the CRC bits appended to the end
 - Continuing our example, $C(x) = x^8 + x^6 + x^5 + x^4 + x + 1$

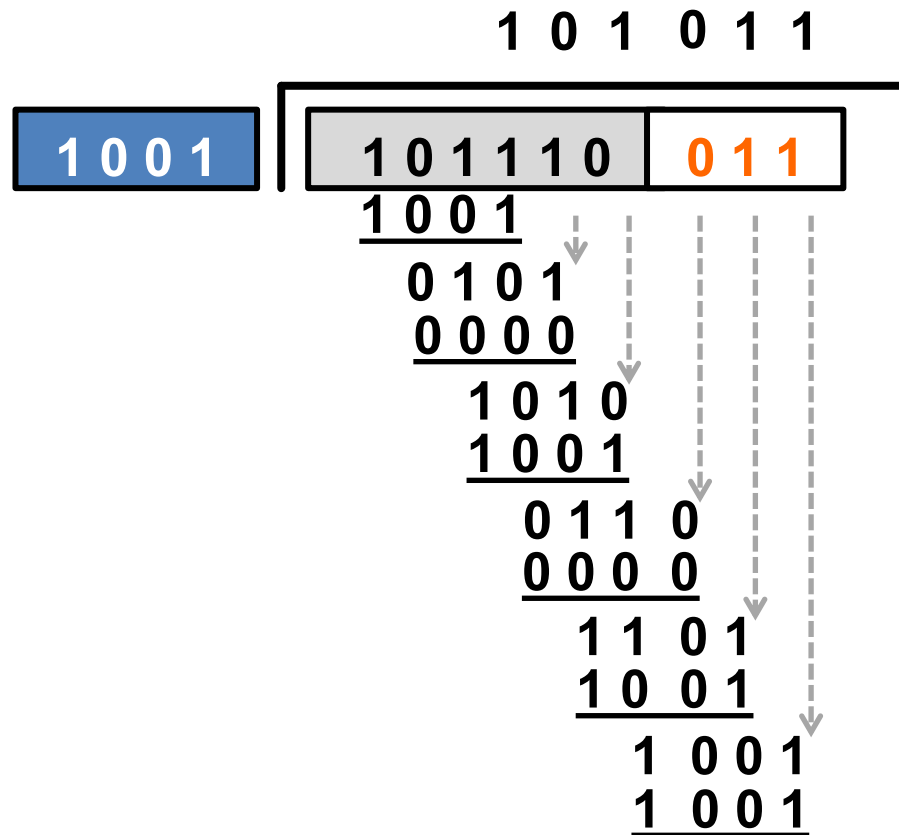
1	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---

Properties of CRC codewords

- Remember: **Remainder** [$T(x)/G(x)$] = $R(x)$
- What happens when we divide $C(x) / G(x)$?
- $C(x) = T(x) + R(x)$ so **remainder** is
 - **Remainder** [$T(x)/G(x)$] = $R(x)$, plus
 - **Remainder** [$R(x)/G(x)$] = $R(x)$
 - Recall, addition is **exclusive-or** operation, so:
 - **Remainder** [$C(x)/G(x)$] = $R(x) + R(x) = 0$

Detecting errors at the receiver

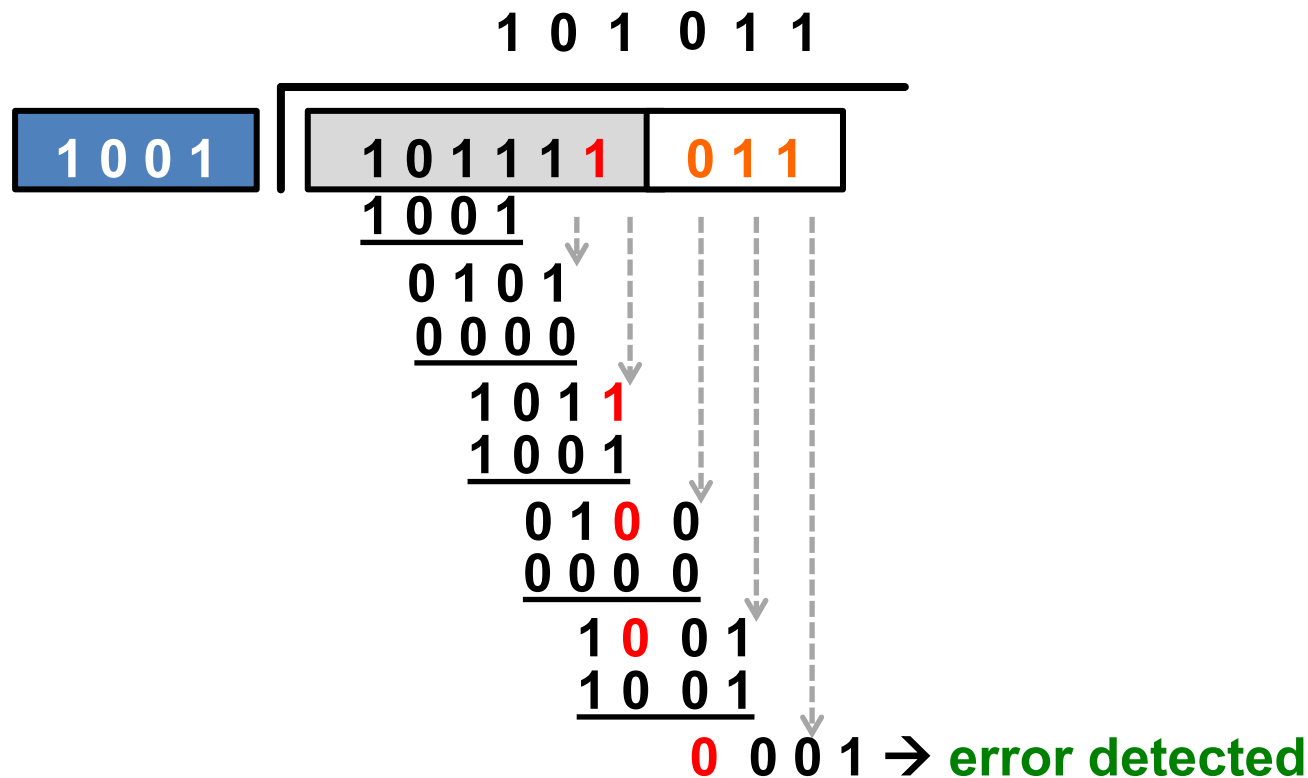
- Divide received message $C'(x)$ by generator $G(x)$
 - If no errors occur, remainder will be zero



0 0 0 → no error detected

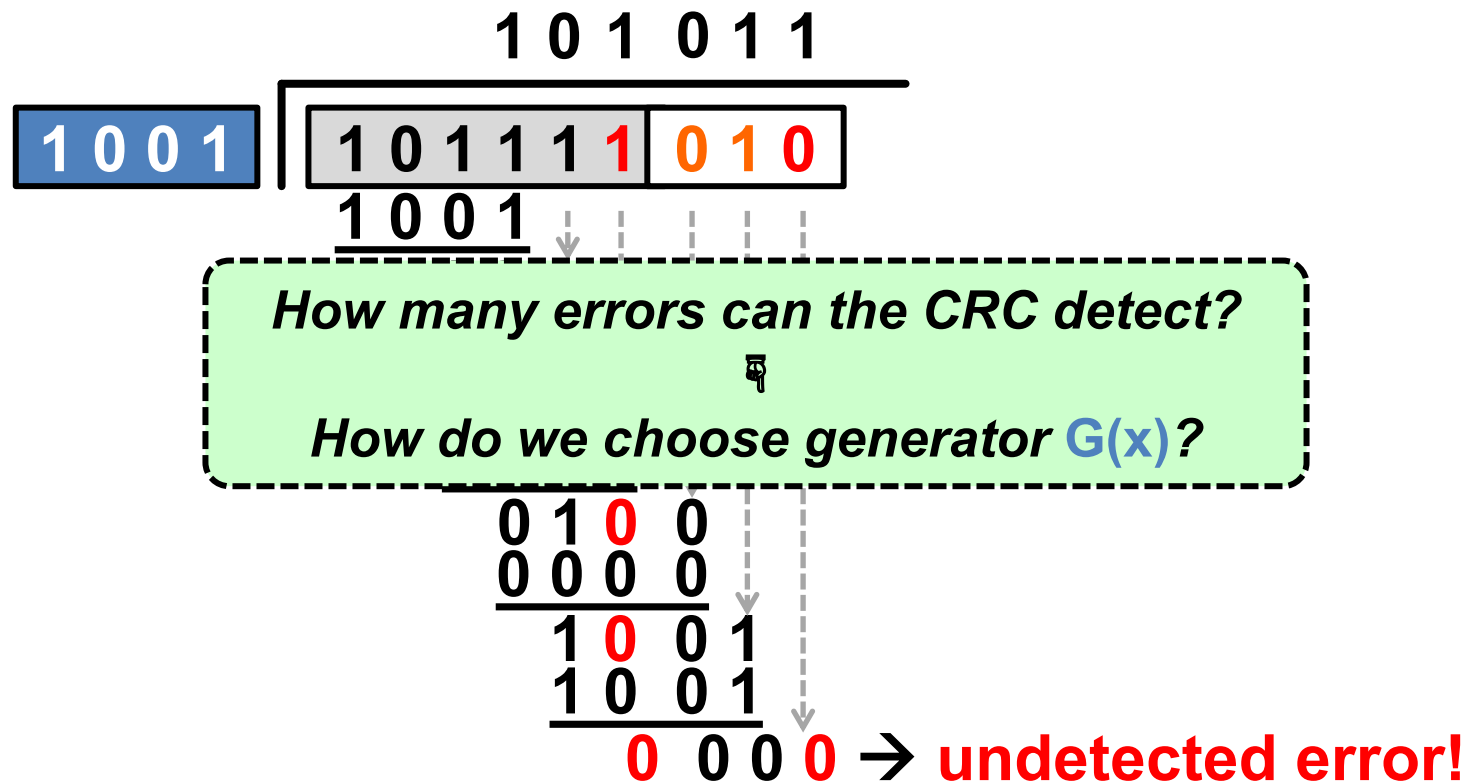
Detecting errors at the receiver

- Divide received message $C'(x)$ by generator $G(x)$
 - If errors occur, remainder **may** be non-zero



Detecting errors at the receiver

- Divide received message $C'(x)$ by generator $G(x)$
 - If errors occur, remainder **may** be non-zero



Detecting errors with the CRC

- The **error polynomial** $E(x) = C(x) + C'(x)$ is the difference between the transmitted and received codeword
 - $E(x)$ tells us which bits the channel flipped
- We can write the **received message $C'(x)$** in terms of $C(x)$ and $E(x)$: $C'(x) = C(x) + E(x)$, so:
 - **Remainder** $[C'(x) / G(x)] = \text{Remainder} [E(x) / G(x)]$
- When does an error go **undetected**?
 - When **Remainder** $[E(x) / G(x)] = 0$

Detecting single-bit errors w/CRC

- Suppose a single-bit error in bit-position i : $E(x) = x^i$
 - Choose $G(x)$ with ≥ 2 non-zero terms: x^{g-1} and 1
 - **Remainder** $[x^i / (x^{g-1} + \dots + 1)] \neq 0$, e.g.:

$$\begin{array}{r}
 \overline{) 001000} \\
 \underline{1001} \\
 1
 \end{array}$$

- Therefore a **CRC with this choice of $G(x)$ always detects single-bit errors** in the received message

Error detecting properties of the CRC

- The CRC will detect:
 - All **single-bit errors**
 - Provided $G(x)$ has two non-zero terms
 - All **burst errors** of length $\leq g - 1$
 - Provided $G(x)$ begins with x^{g-1} and ends with 1
 - Similar argument to previous property
 - All **double-bit errors**
 - With conditions on the frame length and choice of $G(x)$
 - Any **odd number of errors**
 - Provided $G(x)$ contains an even number of non-zero coefficients

Error detecting code: CRC

- Far less overhead than error correcting codes
 - Typically **16 to 32 bits** on a **1,500 byte (12 Kbit) frame**
- **Error detecting** properties are **more complicated**
 - But in practice, “**missed**” bit errors are **exceedingly rare**

Friday Precept:
Work on Lab 2

Tuesday Topic:
Convolutional Codes