# Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

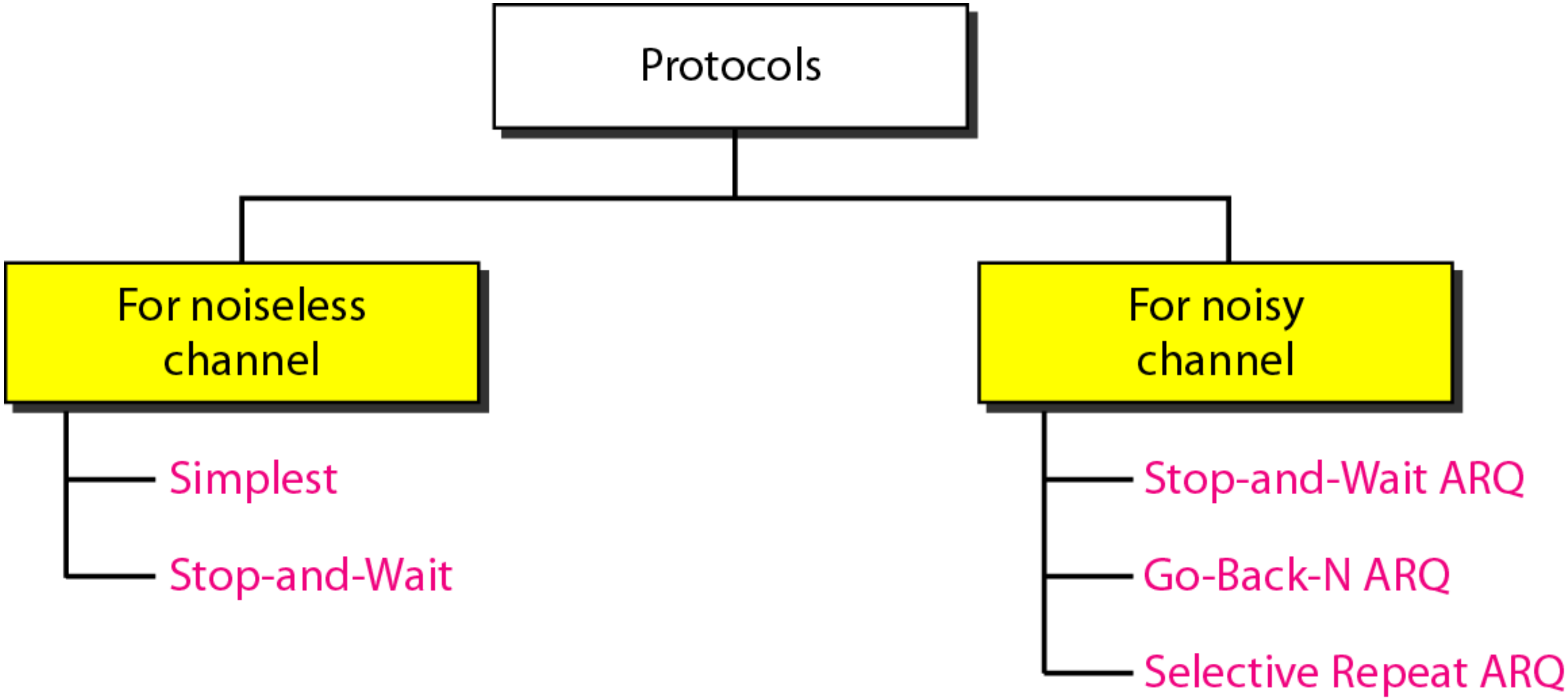
# CSU33031 Computer Networks

## Error Detection

Stefan Weber
email: sweber@tcd.ie
Office: Lloyd 1.41

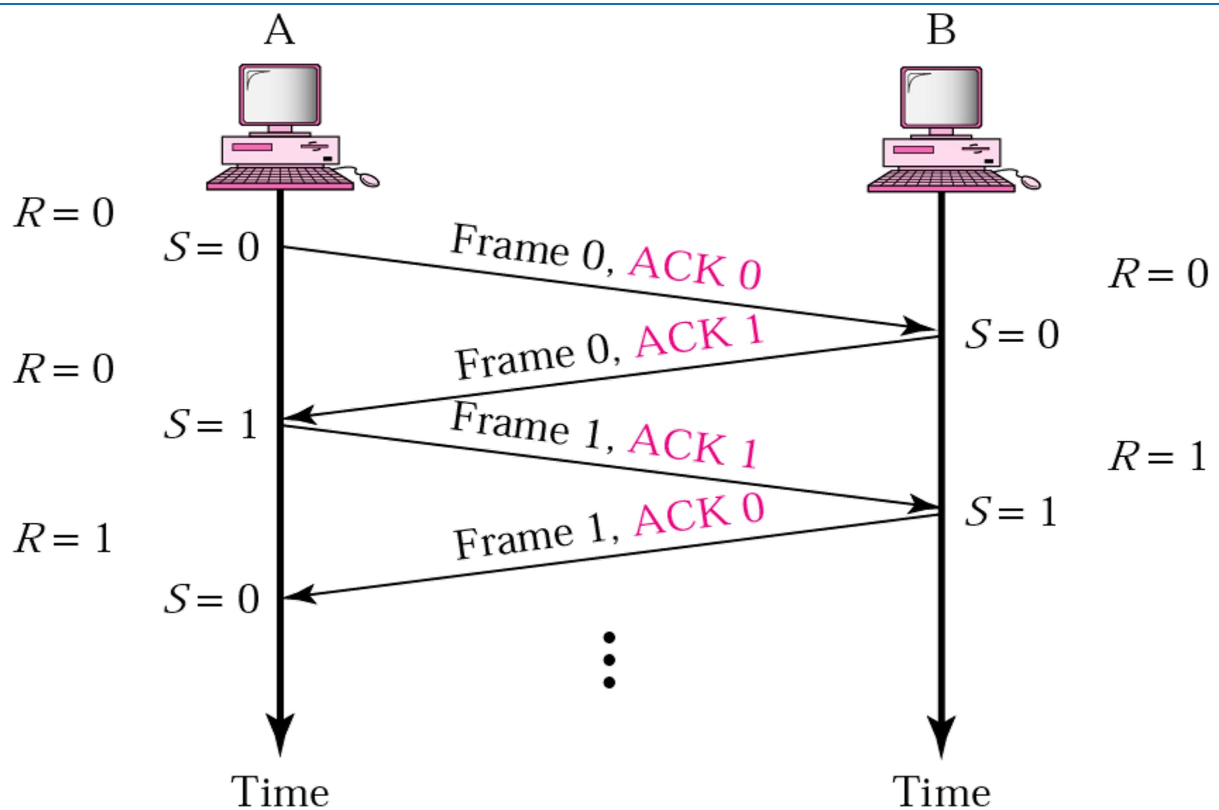
# Review: Flow Control

Flow Control: Refers to the control of the amount of data that a sender can transmit **without overflowing the receiver**.




* Figure is courtesy of B. Forouzan

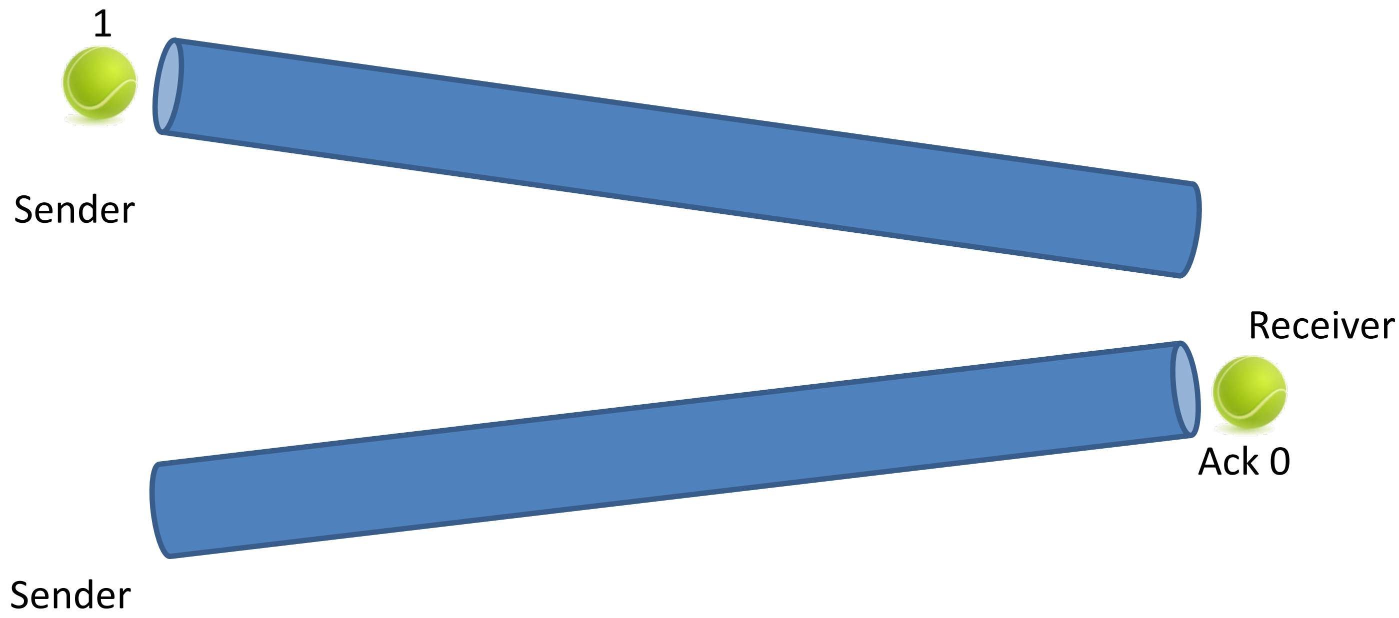
# Stop-and-Wait ARQ



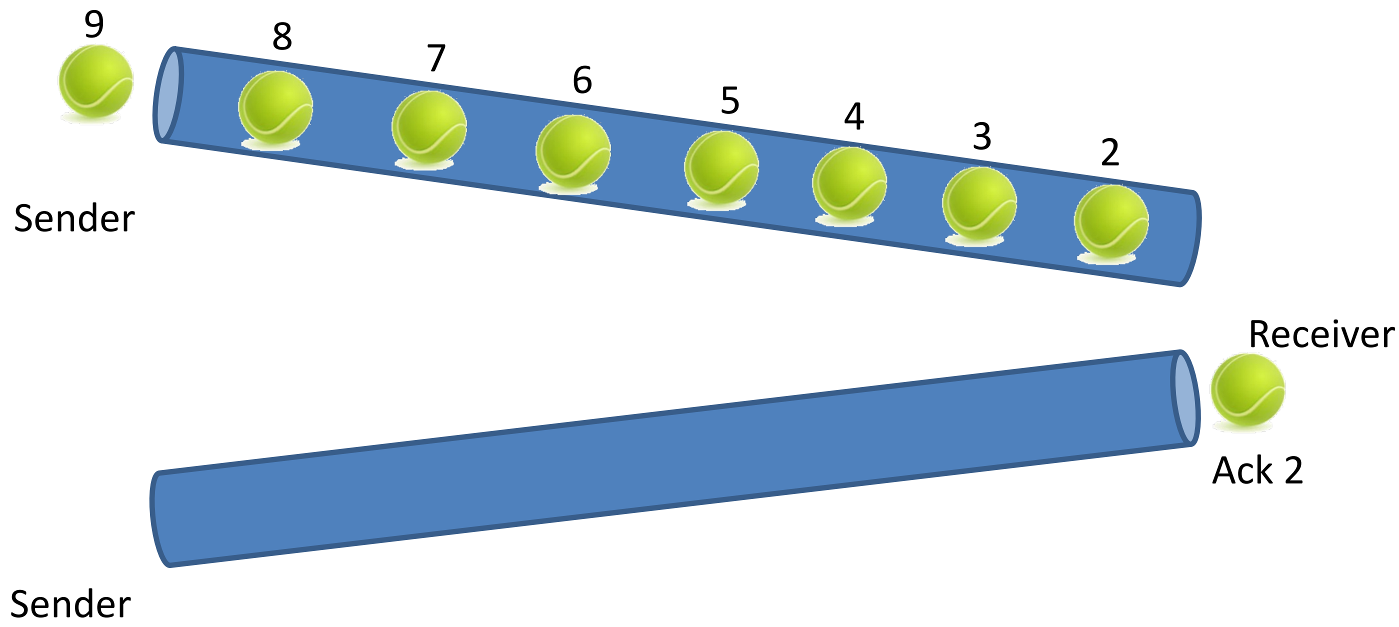

- ACK = received packet, ready to receive packet #
- Next data frame send carries acknowledgement for last frame received

* Figure is courtesy of B. Forouzan

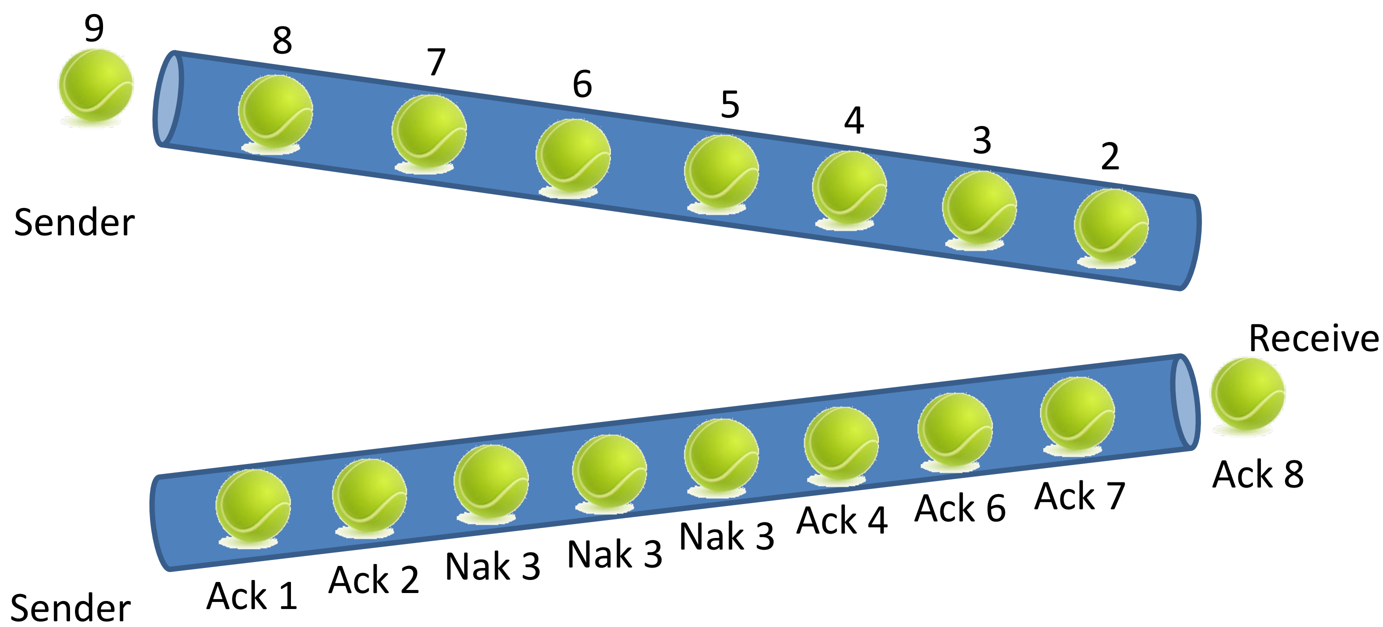
# Bandwidth-Delay Product: Example



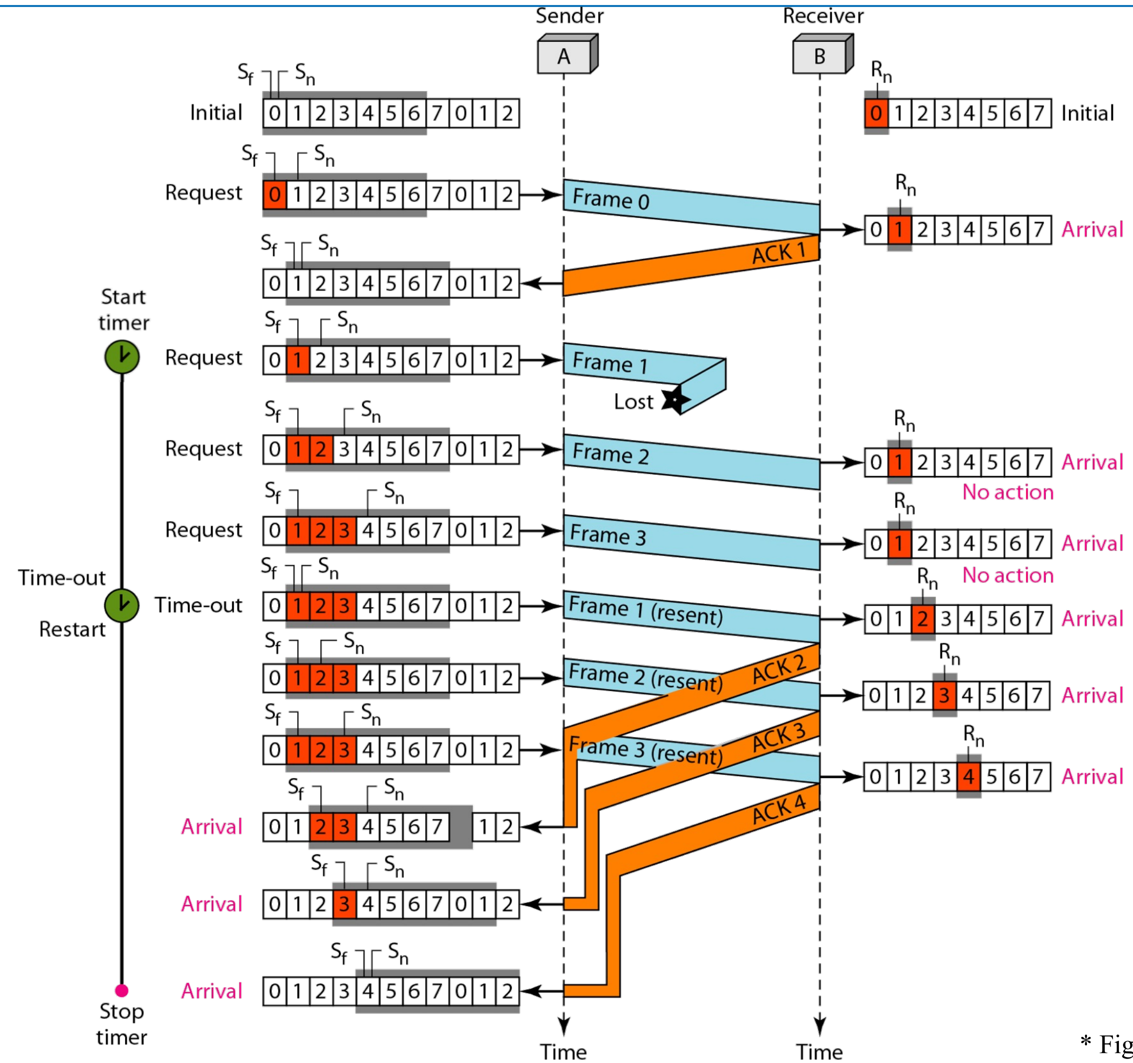
# Bandwidth-Delay Product: Example



# Bandwidth-Delay Product: Example

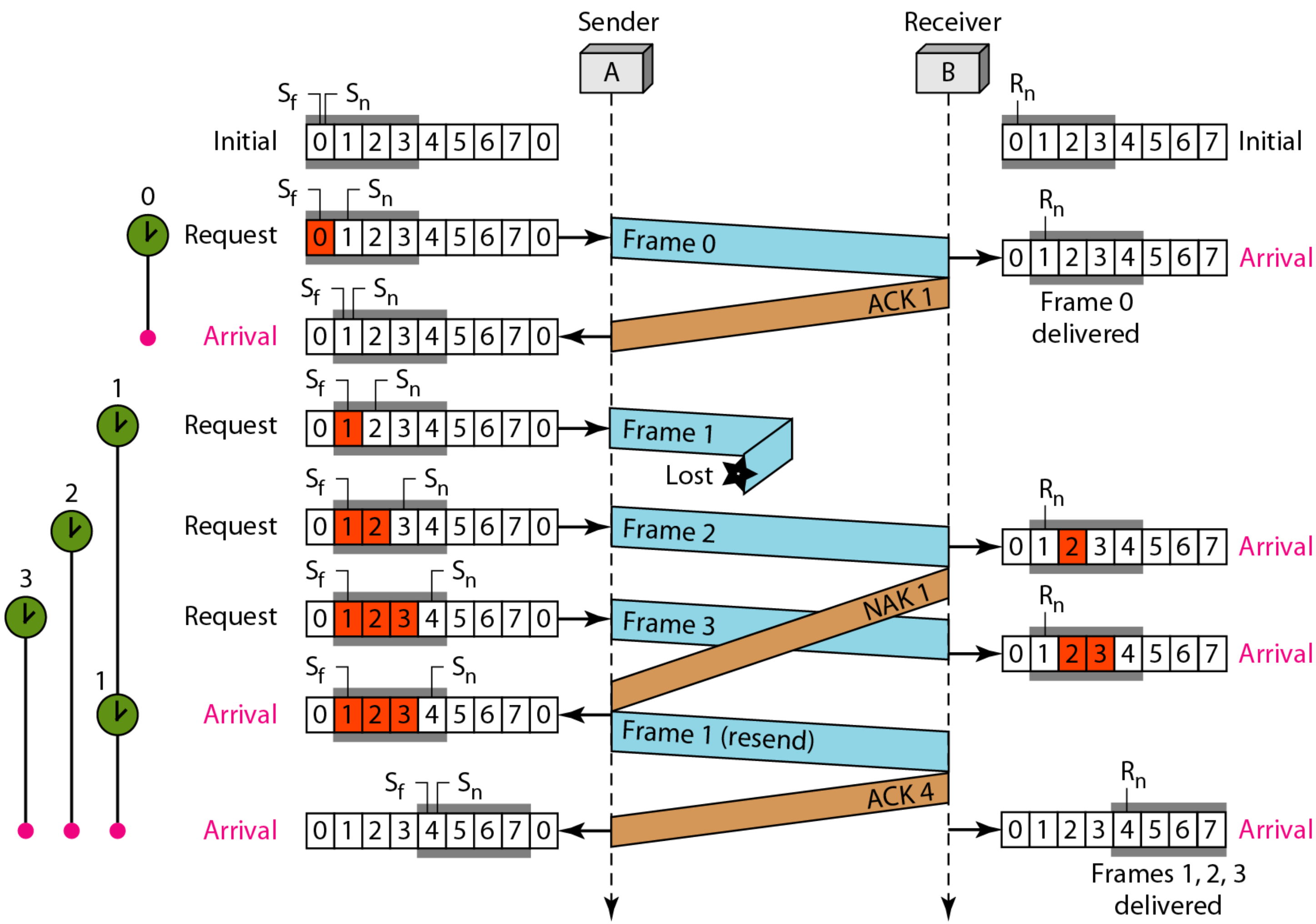


# Go-Back-N ARQ: Bad Behaviour




* Figure is courtesy of B. Forouzan

# Selective Repeat ARQ

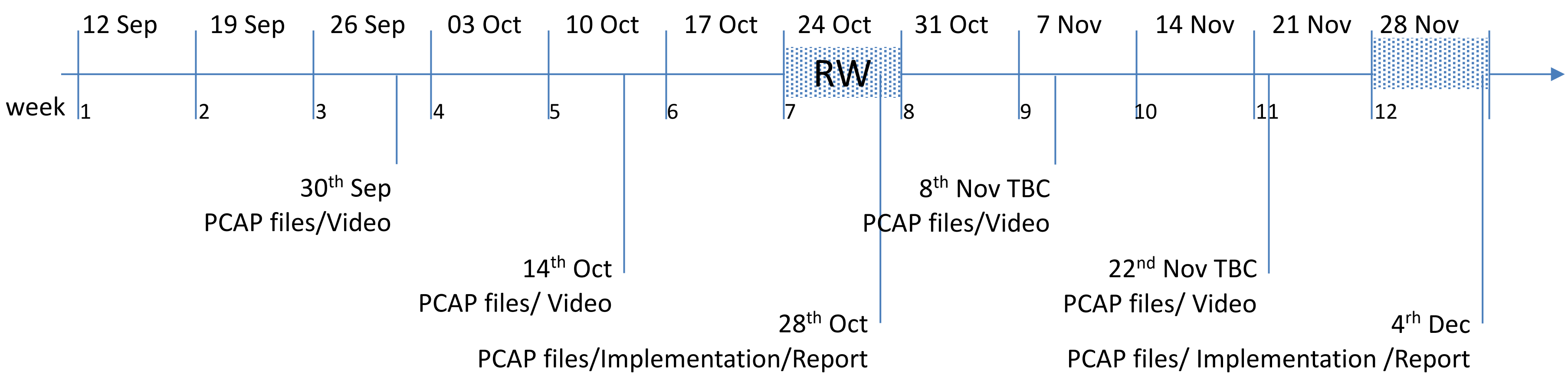



* Figure is courtesy of B. Forouzan

# Window Size for Go-Back-N, etc

- Depends on size of max. frame number
  - Frame # needs to be included in every frame
  - e.g. m = 4 bits –> $2^4 = 16$ frame numbers
  - GoBackN window size = $2^m-1$ eg. 15
  - Selective Repeat window size = $2^{m-1}$ eg. 8
- Trade-off between window size and header size

# CSU33031 Timeline & Weights




- 60% Coursework
  - 30% Assignment 1
  - 30% Assignment 2
- 40% Exam
- Supplemental exam - 100%

# Assignment 1

The goal of the assignment is for a client to send a request for a file to a server, called ingress in the example. The server will then distribute the request to one of the workers that it knows of - either by having hard-coded references or by having the worker register with the server when they start. When a worker receives a request for a file, it will load this file and return it to the server, and the server in turn will return it to the client.

# Assignment 1

The easiest way to start with the development of your solution is possibly to connect your components through the localhost interface of a machine; however, at the end, you will need to be able to demonstrate that your protocol can connect components located at a number of hosts. There are a number of platforms that support the simulation of topologies or provide virtual infrastructures e.g. Docker [2], Mininet [6], Kubernetes [1], etc.

# Assignment 1

For someone starting with socket programming and networking, I would suggest to use a platform such as Docker or Mininet; for someone already familiar with these concepts, I would suggest to implement their solution using Kubernetes. However, these are only suggestions and you need to make the decision how to implement your solution. The use of Docker for the assignments is optional – All assignments can be implemented and run on personal laptops or computers in the labs as well. No one will be penalised for not using Docker for the assignments.

# Assignment 1

The video of part 1 should demonstrate the initial design of your solution and a capture of network traffic between the components of your solution and the files of your traffic capture in pcap format. In the video, you should explain the setup of the topology that you are using and the information that makes up the header information in your traffic captures. The submission process for this part consists of two steps:
1) Submitting the PCAP file or files that you captured from your network traffic, and 2) submitting the video for this step.

# Assignment 1

The video is to be no longer than 4 minutes; content past the 4 minute-mark will be ignored during marking. Videos with voice-over using text-to-speech (TTS) will not be accepted and marked with 0.

The video should describe your progress in the first couple of weeks. I think of these videos as a replacement of an update in a standup meeting where you have been given the task to implement this protocol and during a regular meeting you need update your team on your progress i.e. what you have done, what your thoughts are behind your decisions and then what you plan is going forward.

# Assignment 1

A rough guidance for marking the initial videos last year was as follow:
1 point - for providing a video
2 points - the before + and demonstrating a traffic capture
3 points - the before + an explanation of network traffic
4 points - the before + demonstrating that they have started on their own protocol
5 points - the before + demonstrating their own protocol on virtual infrastructure

There are a number of sample videos in the Assignment 1 folder in Blackboard.

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

# CSU33031 Computer Networks

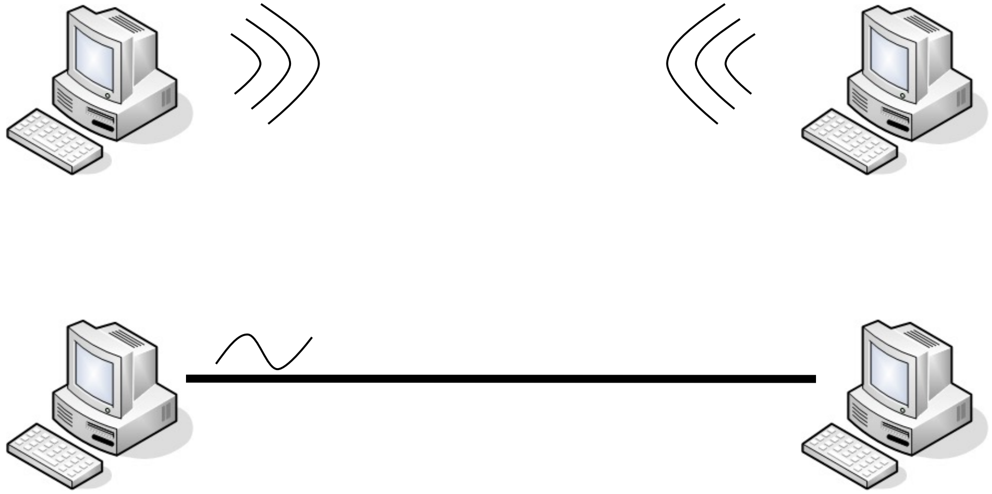
## Error Detection and Correction

Stefan Weber
email: sweber@tcd.ie
Office: Lloyd 1.41

# Errors in Transmissions

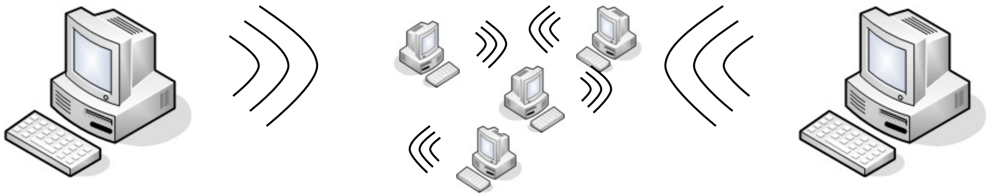
- Causes for Errors
- Types of Errors
- Detection of Errors
- Correction of Errors

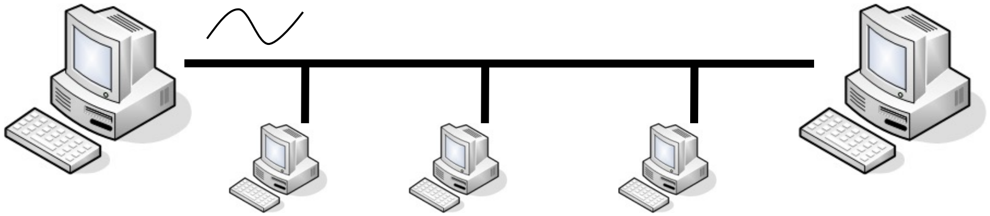
# Terminal to Terminal Comms



- Either over dedicated or shared medium

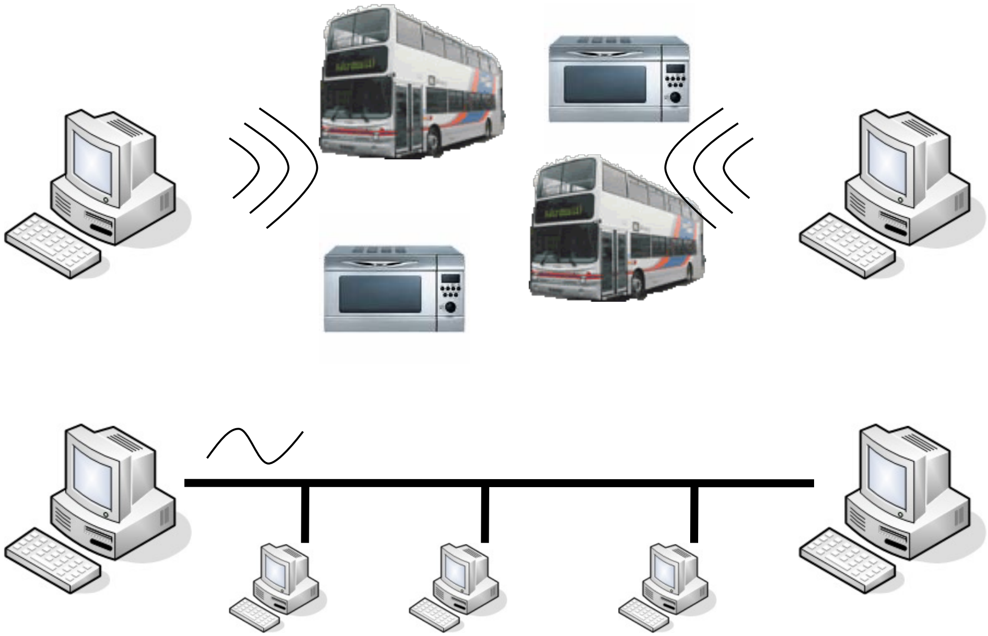
# Causes for Errors





- Interference
  - Collision with communication from other nodes
  - Electrical interference from third parties
  - Thermal interference

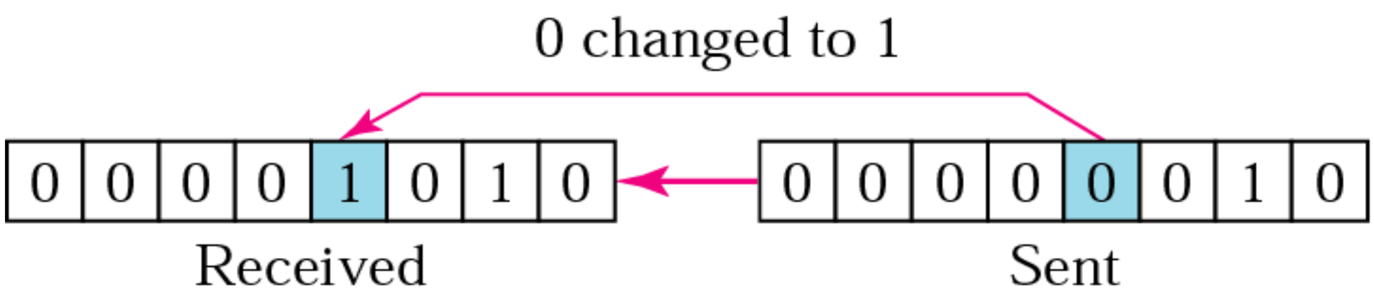
# Causes for Errors



- Interference
  - Collision with communication from other nodes
  - Electrical interference from third parties
  - Thermal interference

# Types of Errors: Single-Bit Error

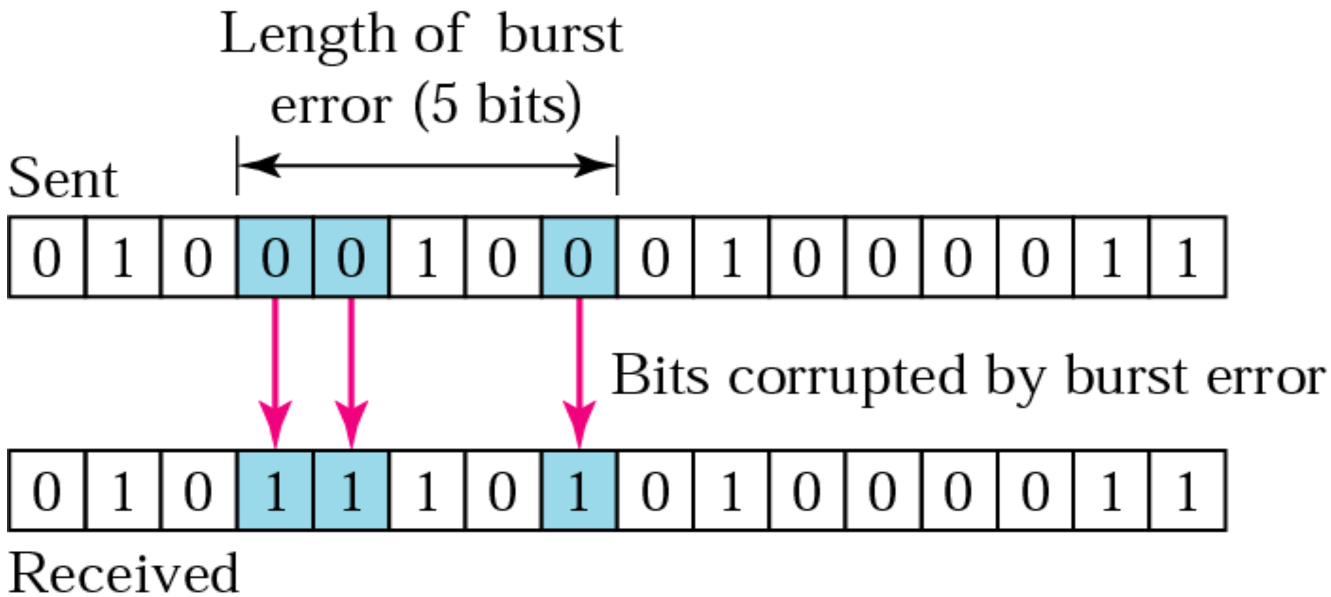
In a single-bit error, only one bit in the data unit has changed.




* Figure is courtesy of B. Forouzan

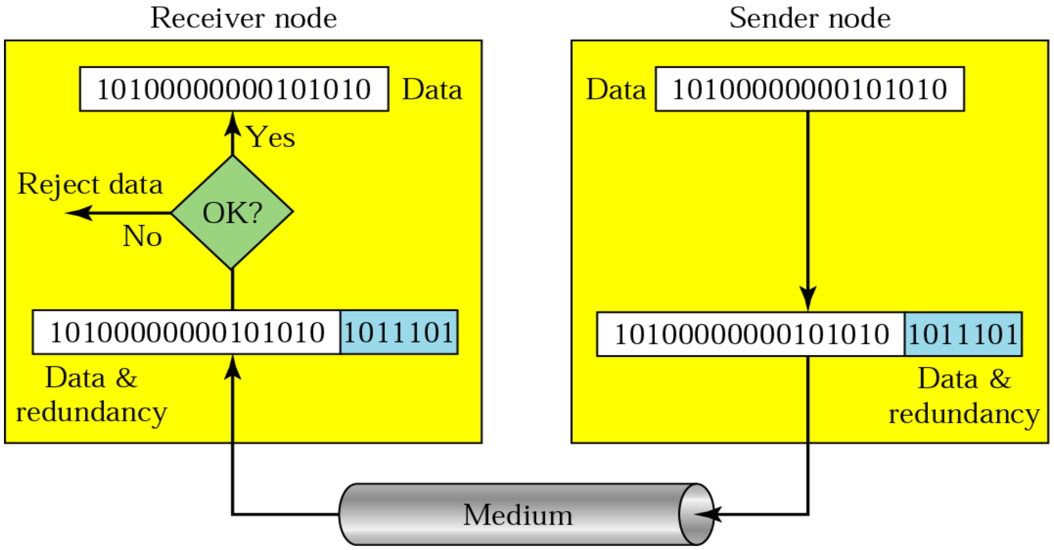
# Types of Errors: Burst Error

A burst error means that 2 or more bits in the data unit have changed




* Figure is courtesy of B. Forouzan

# Redundancy



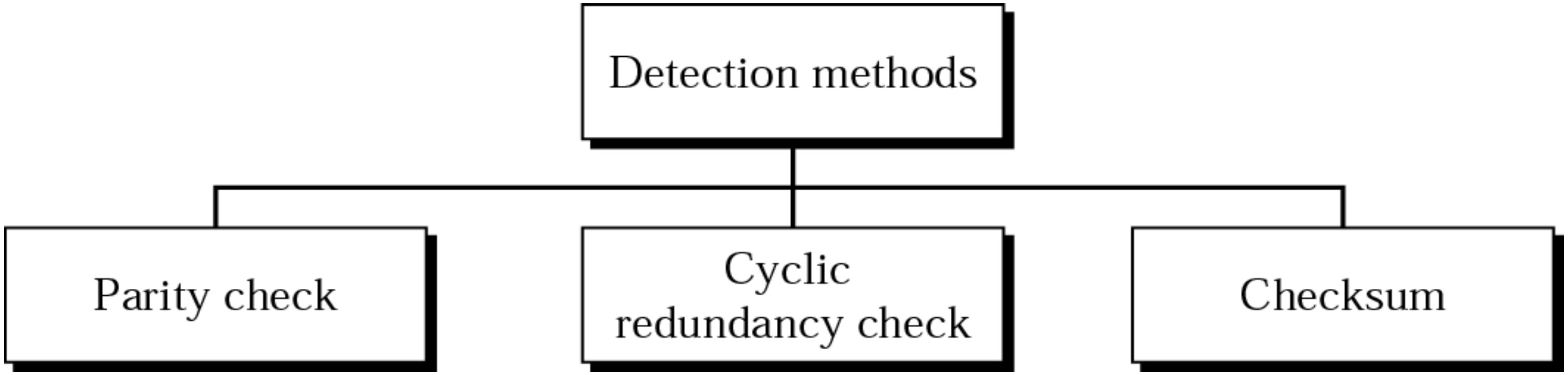

– Sender  includes additional information

– Receiver verifies this information

– Example: Meet Thursday, 26th Sep    (→ Wrong!)

* Figure is courtesy of B. Forouzan

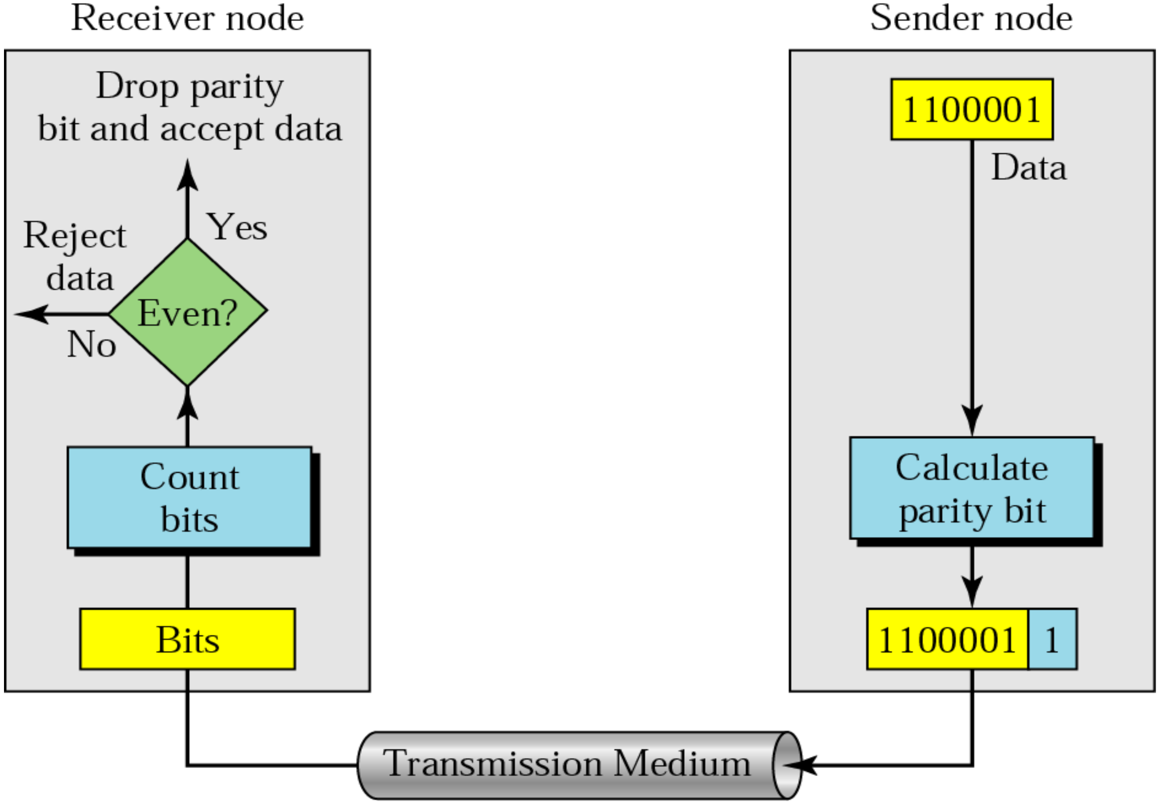
# Detection of Errors




- Types of detection methods
  - Balance **detection against overhead**

* Figure is courtesy of B. Forouzan

# Even-Parity Concept




A parity bit is added to every data unit so that the total number of 1s is even (or odd for odd-parity).

* Figure is courtesy of B. Forouzan

# Even-Parity: Example - Sender

- Assume you want to send the following:

1110111 1101111 1110010 1101100 1100100

- The following bits are actually sent:

1110111**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

# Even-Parity: Example - Receiver

11101110 11011110 11100100 11011000 11001001

6 6 4 4 4

- The receiver counts the 1s in each character and comes up with even numbers (6, 6, 4, 4, 4). The data are accepted.

---

11111110 11011110 11101100 11011000 11001001

7 6 5 4 4

- The receiver counts the 1s in each character and comes up with even and odd numbers (7, 6, 5, 4, 4). The receiver knows that the data are corrupted, discards them, and asks for retransmission.

# Simple Parity Check

- Can detect all single-bit errors

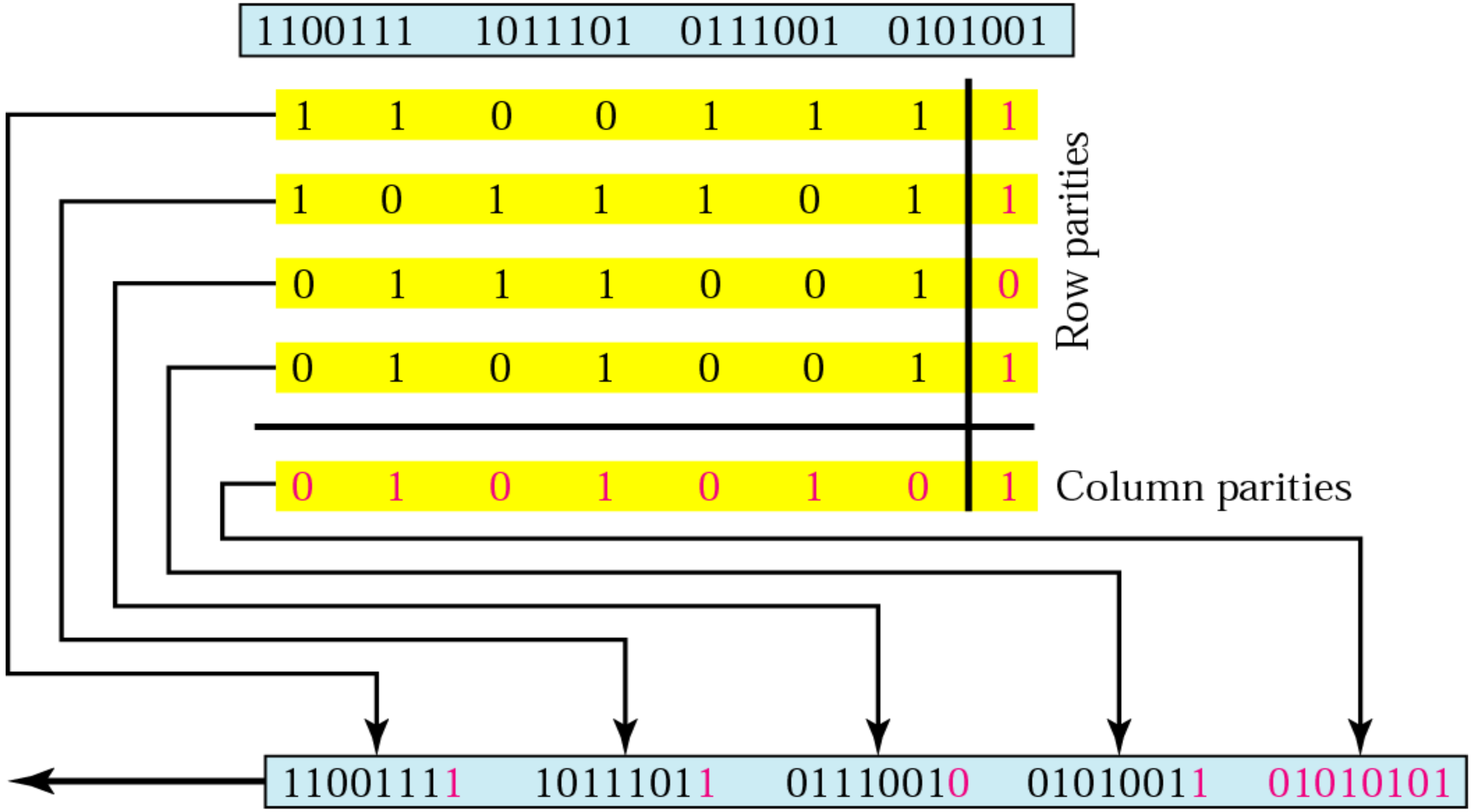
- Can detect burst errors only if the total number of errors in each data unit is odd

- Overhead: 7000 bits of data require

  1000 bits of redundant info.

# Two-Dimensional Parity Check

In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.




* Figure is courtesy of B. Forouzan

# Example: 2D-Parity Check

Suppose the following block is sent:

10101001   00111001   11011101   11100111   10101010

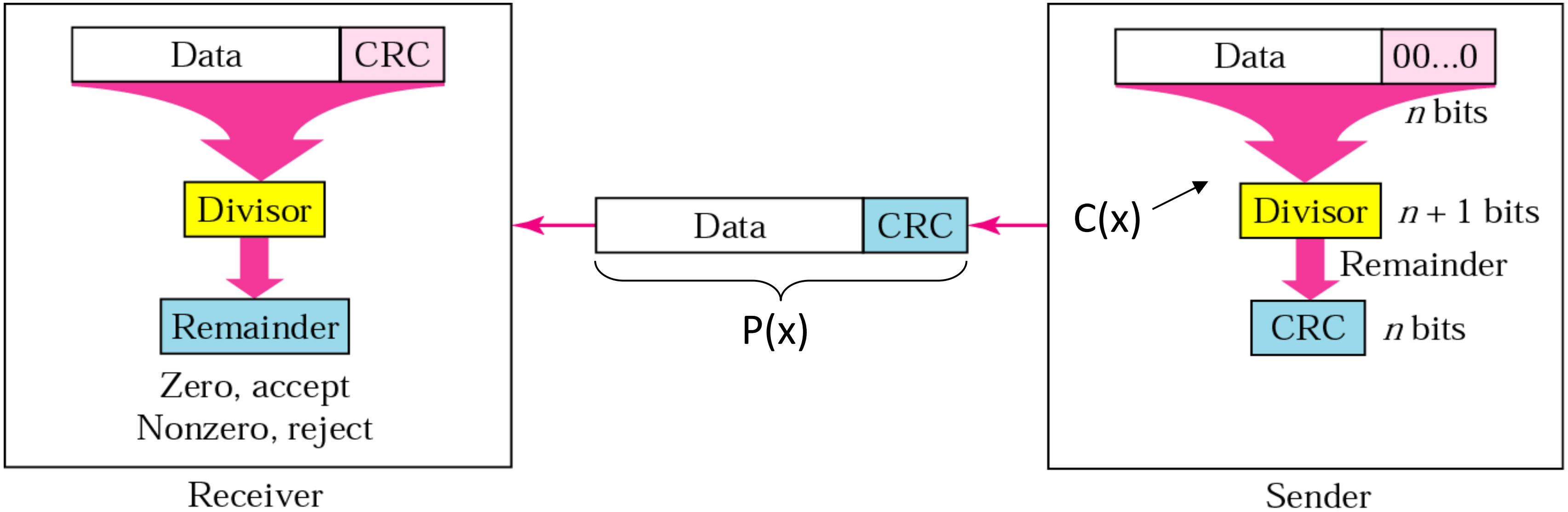
However, it is hit by a burst noise of length 8, and some bits are corrupted.

1010**0011**   **1000**1001   11011101   11100111   10101010

When the receiver checks the parity bits, some of the bits do not follow the even-parity rule and the whole block is discarded.

10100011   10001001   11011101   11100111   **1**0**101**0**1**0

# Cyclic Redundancy Check (CRC)




- P(x) divided by C(x) = 0
- (P(x)+remainder) divided by C(x) should be != 0

* Figure is courtesy of B. Forouzan

# Division – Decimal & Binary

39 / 20 = 1 + 19

100111 / 10100 = 1 + 10011

32 4 2 1 16 4 16 2 1

# CRC Calculation

- CRC Calculation→ Polynomial Division

  not Binary Division!!!

$$
\begin{array}{l}
x^3 + 4x^2 + 3x + 12 \quad / \quad x^2 + 3 \quad = \quad x + 4 \\
x^3 \qquad\quad + 3x \\
\qquad 4x^2 \qquad\quad + 12 \\
\qquad\qquad\qquad \text{-----} \\
\qquad\qquad\qquad\quad 0
\end{array}
$$

# CRC Calculation

- CRC Calculation→ Polynomial Division

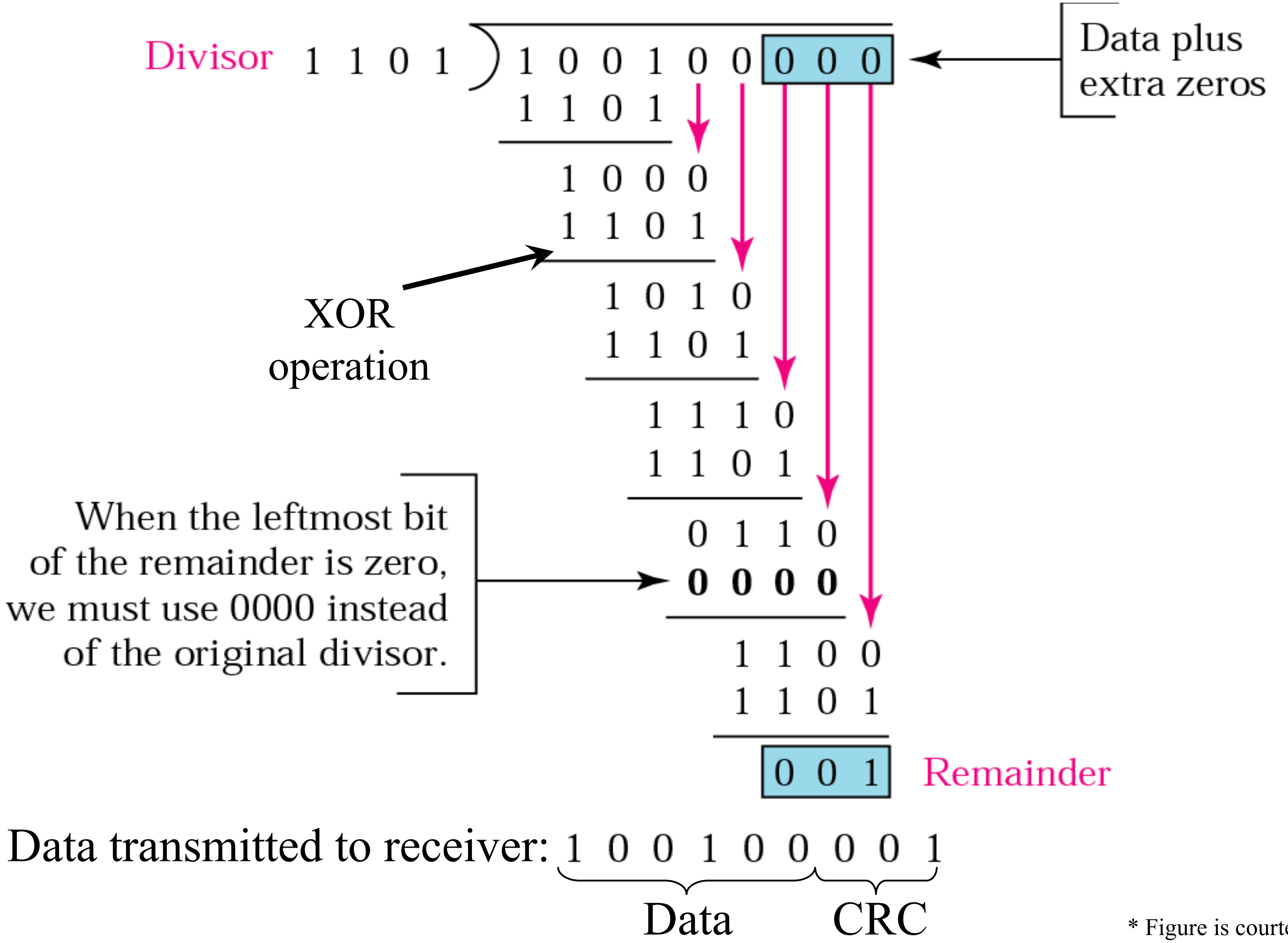
  not Binary Division!!!

- CRC: Coefficient r={0,1}

100001000000000001011 /    10001000000100001

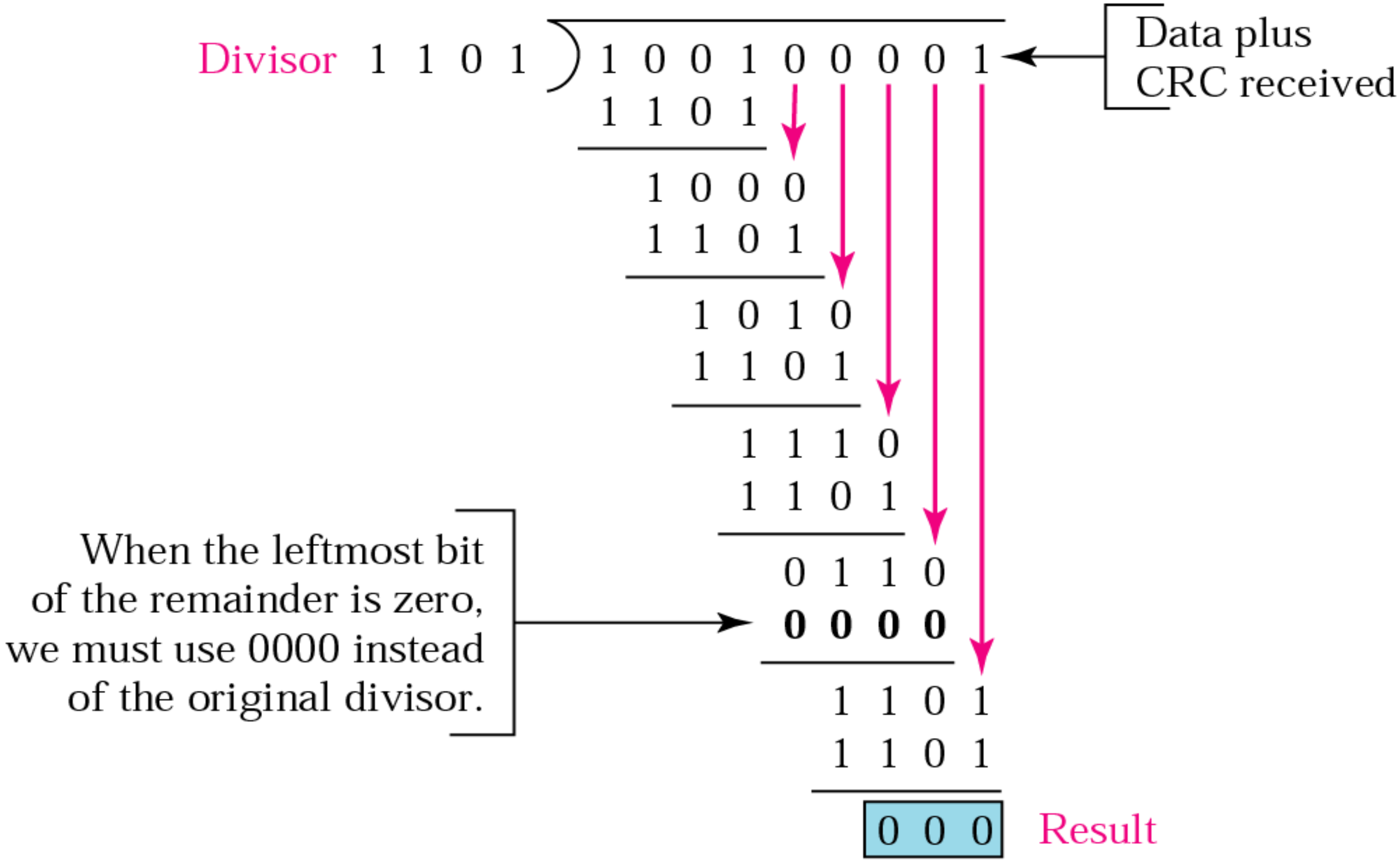
$$x^{20} + x^{15} + x^4 + x + 1 \quad / \quad x^{16} + x^{12} + x^5 + 1$$

# CRC: Sender



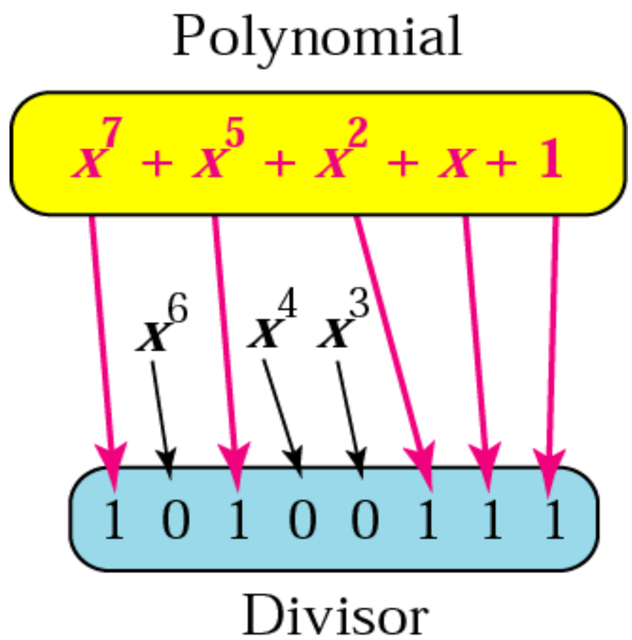

* Figure is courtesy of B. Forouzan

# CRC: Receiver




* Figure is courtesy of B. Forouzan

# Polynomial Notation




- Rules for selecting divisor:
  - It should not be divisible by x
  - It should be divisible by x+1

* Figure is courtesy of B. Forouzan

# Polynomials

- We cannot choose **x** (binary 10) or $\mathbf{x^2 + x}$ (binary 110) as polynomial because both are divisible by x.
- However, we can choose **x + 1** (binary 11) because it is not divisible by x, but is divisible by **x + 1**. We can also choose $\mathbf{x^2 + 1}$ (binary 101) because it is divisible by **x + 1** (binary division).

* Figure is courtesy of B. Forouzan

# Standard Polynomials

| Name | Polynomial | Application |
|---|---|---|
| CRC-8 | $x^8 + x^2 + x + 1$ | ATM header |
| CRC-10 | $x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ | ATM AAL |
| CRC-16 | $x^{16} + x^{12} + x^5 + 1$ | HDLC |
| CRC-32 | $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ | LANs |

* Figure is courtesy of B. Forouzan

# CRC Performance

- Can detect all burst errors that effect an odd number of bits
- Can detect all burst errors of the length less than or equal to the degree of the polynomial
- Can detect with a very high probability burst errors of a length greater than the degree of the polynomial.

# CRC-12 Example

The CRC-12

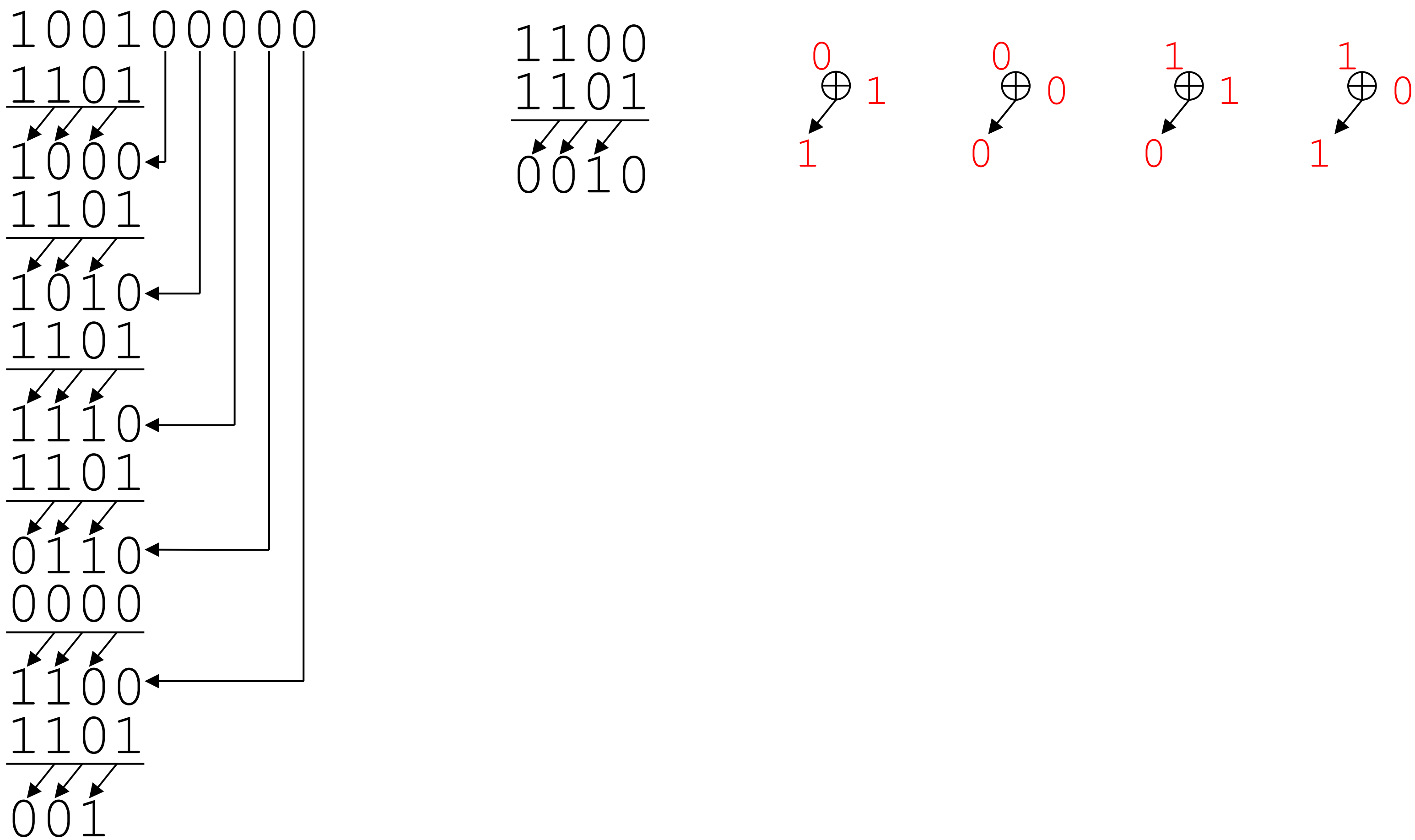
$$x^{12} + x^{11} + x^3 + x + 1$$

which has a degree of 12, will detect all burst errors affecting an odd number of bits, will detect all burst errors with a length less than or equal to 12, and will detect, 99.97 percent of the time, burst errors with a length of 12 or more.

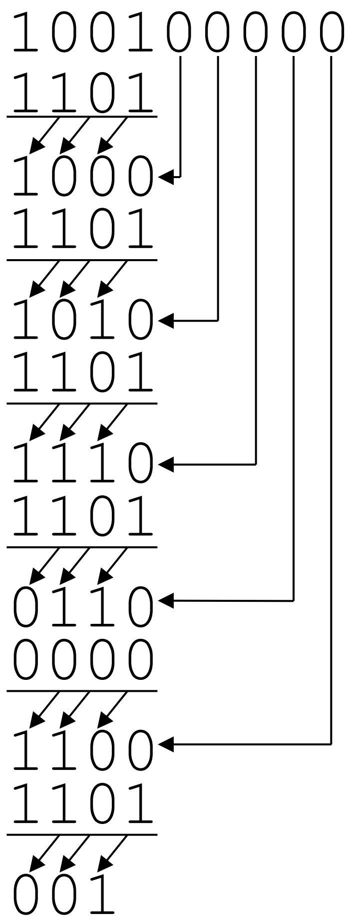
# CRC Calculation

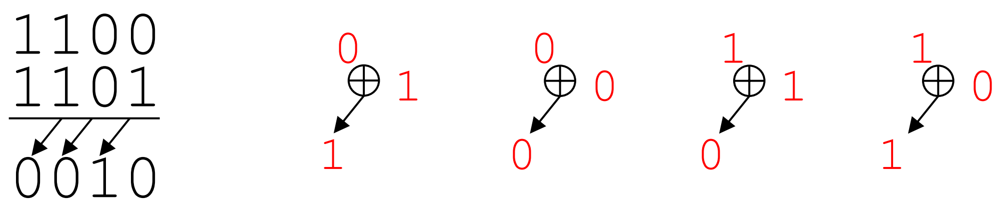
```
100100000
1101
----
1000
1101
----
1010
1101
----
1110
1101
----
0110
0000
----
1100
1101
----
001
```

# CRC Calculation



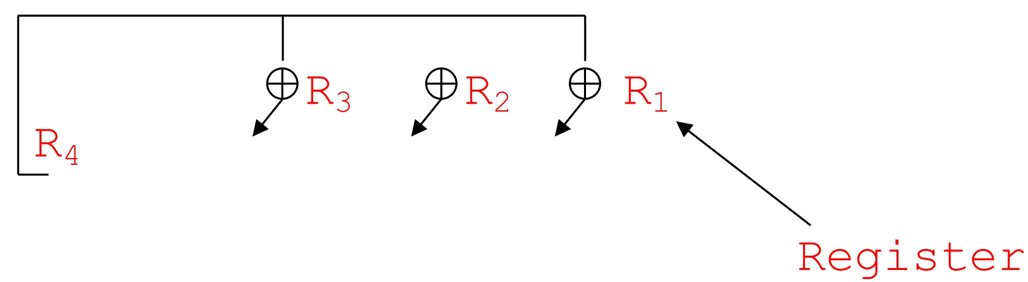
# CRC Calculation



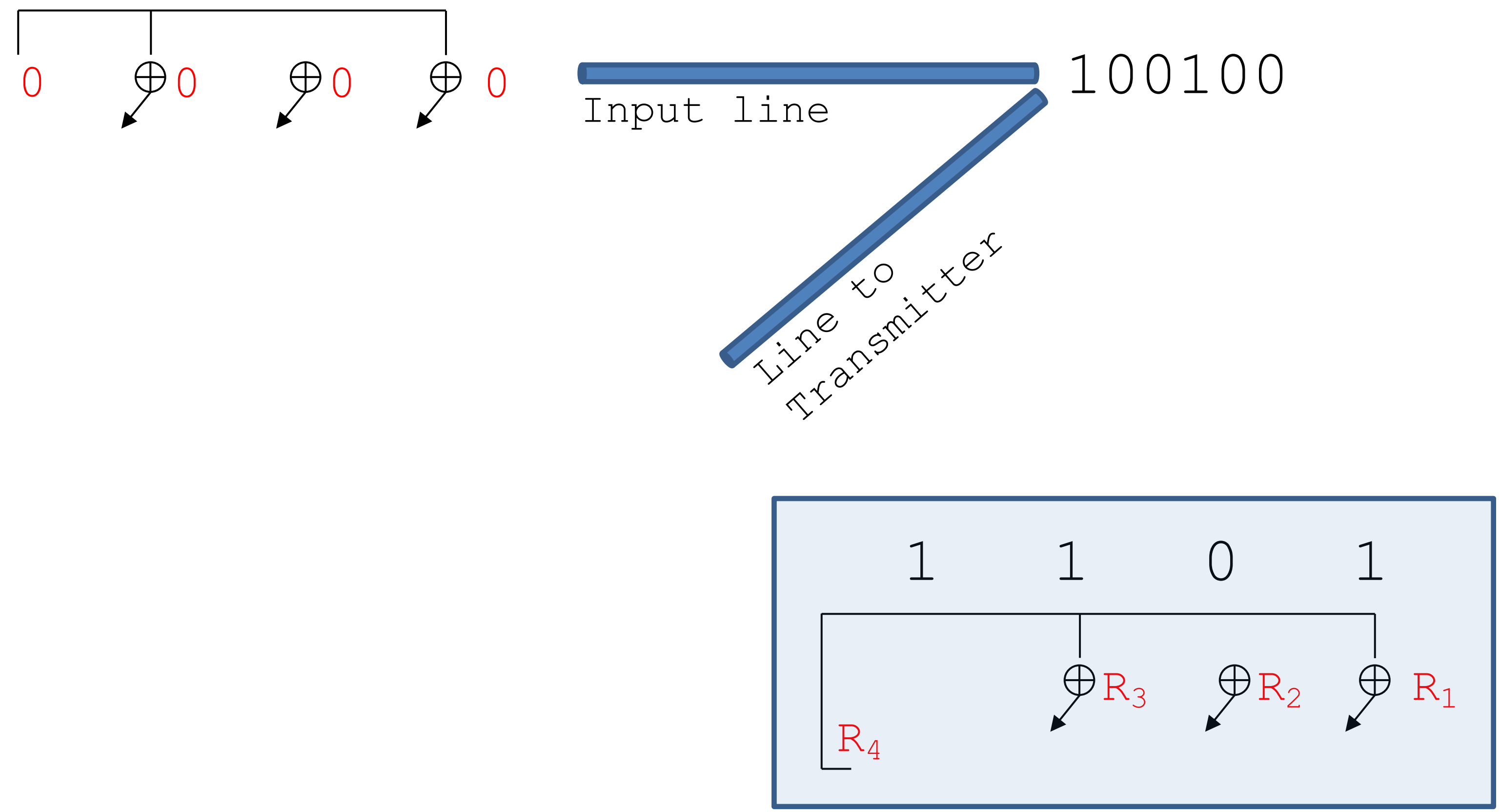




Representation of divisor:

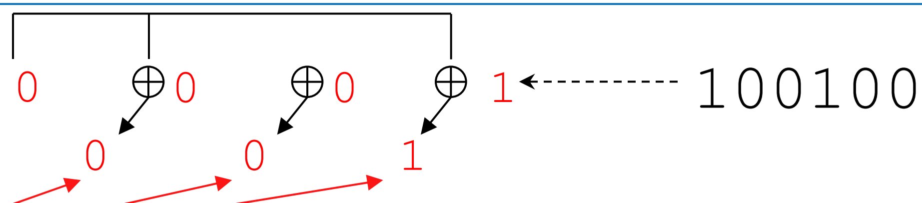
1 1 0 1



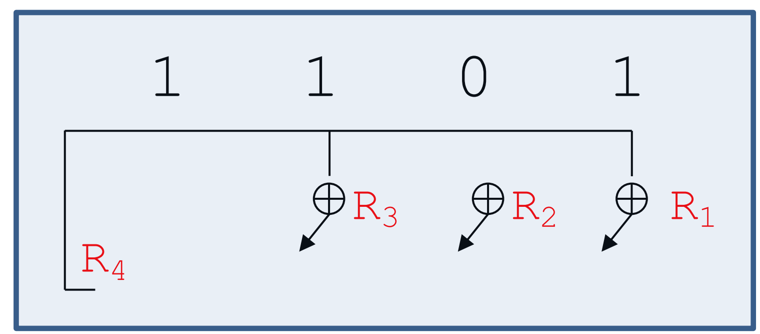
# CRC Calculation



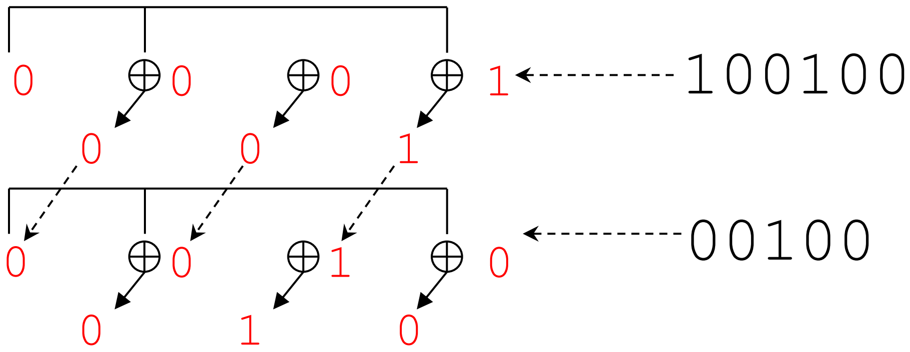
# CRC Calculation



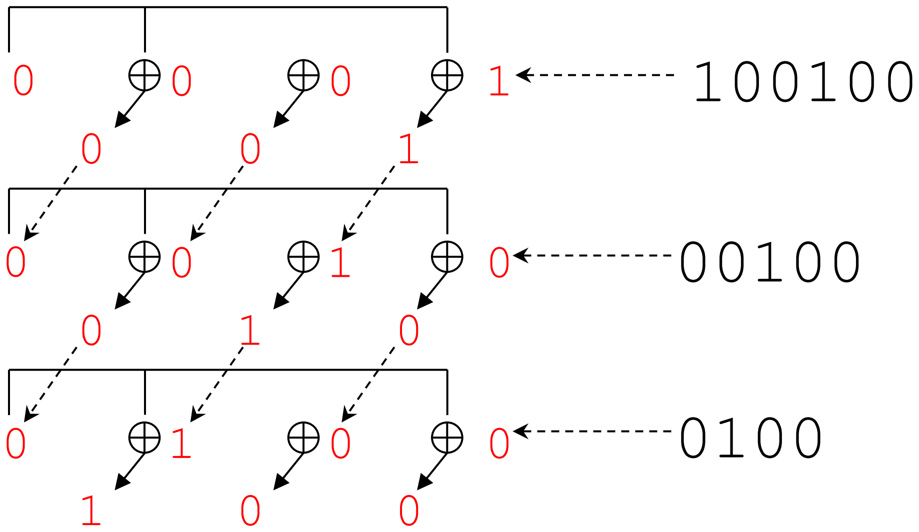

New content
for registers
$R_4$, $R_3$, $R_2$



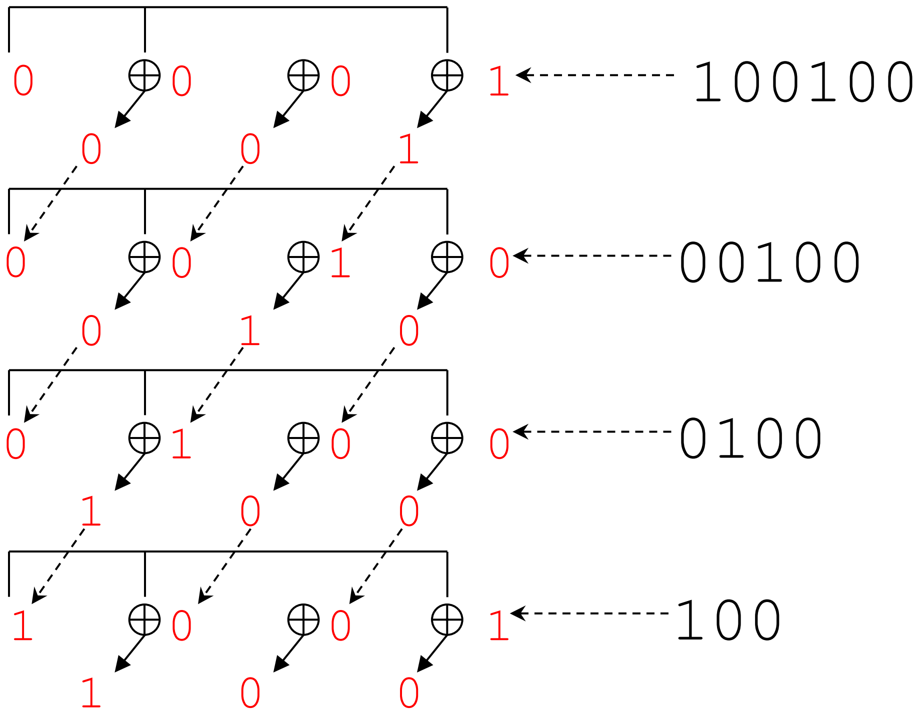
# CRC Calculation



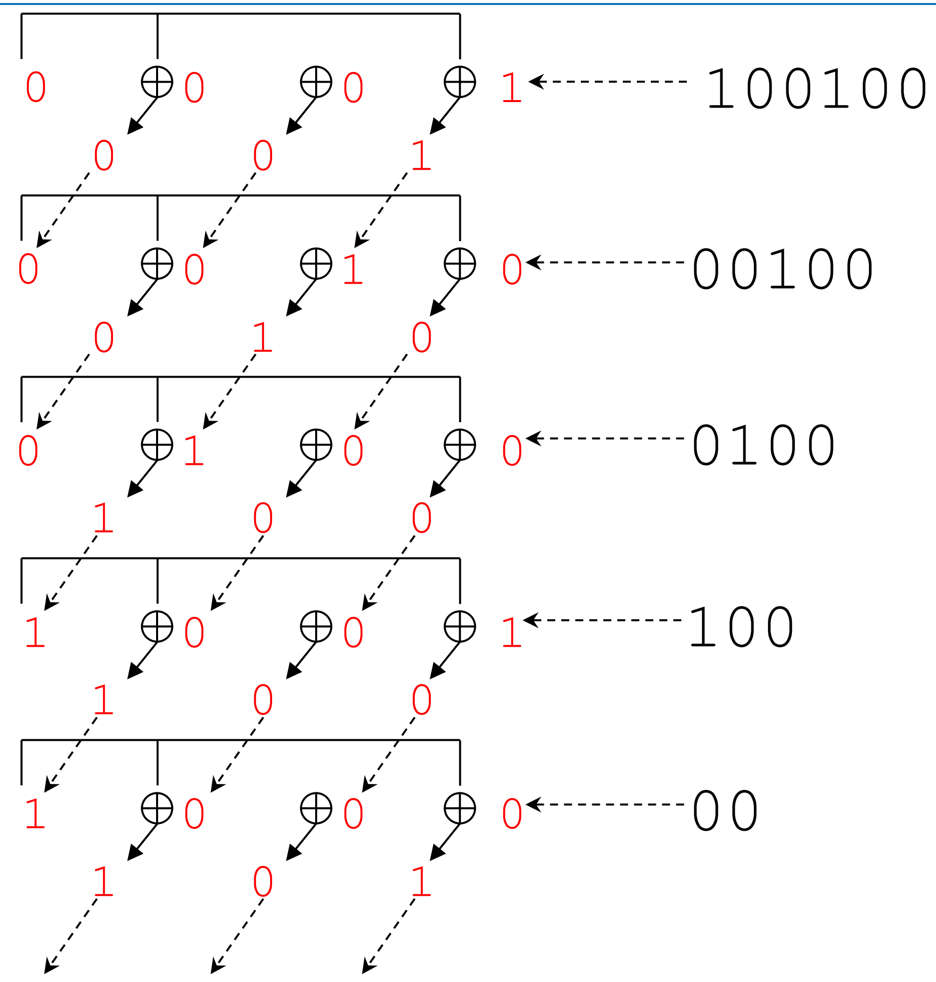
# CRC Calculation



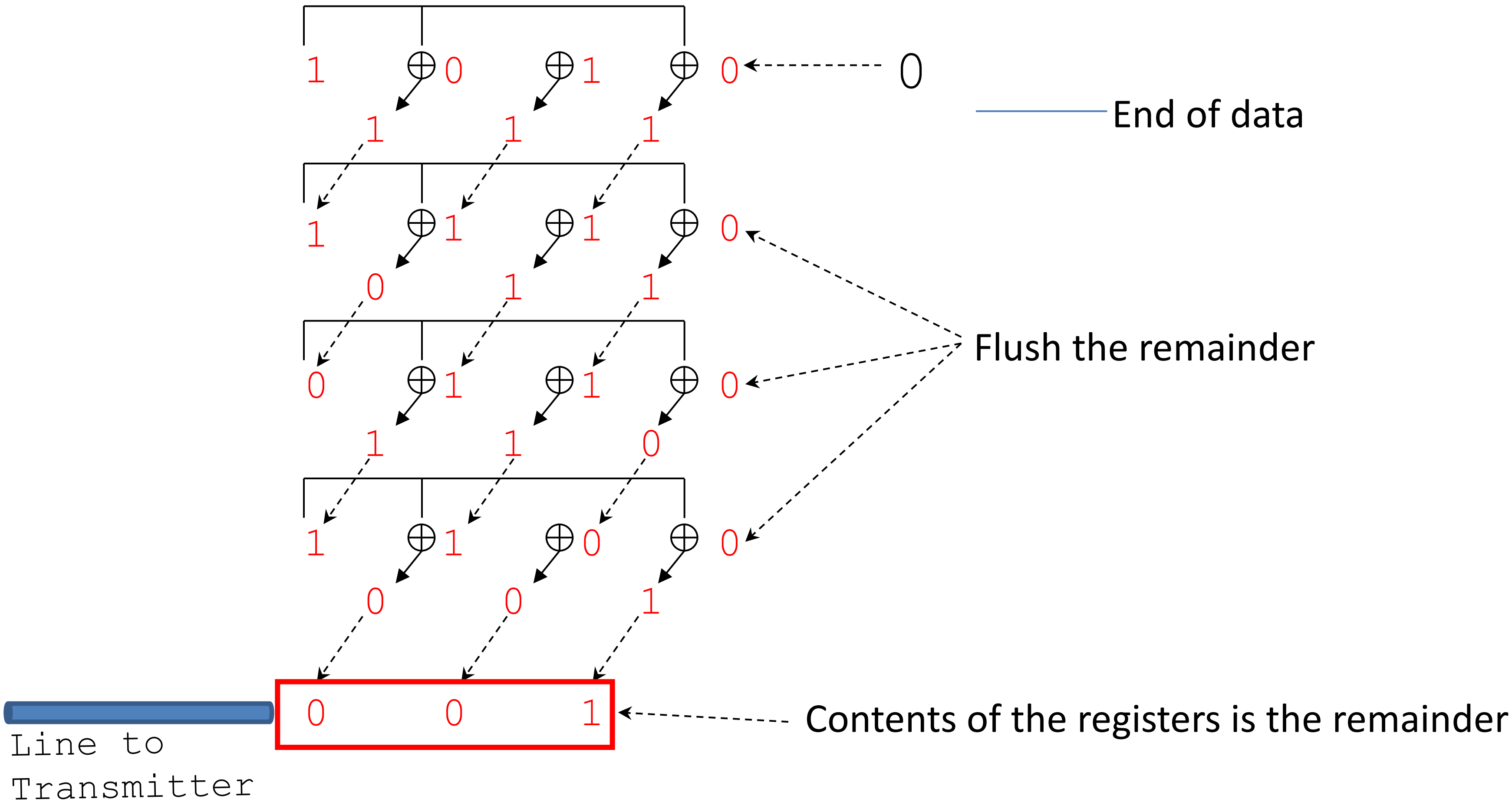
# CRC Calculation



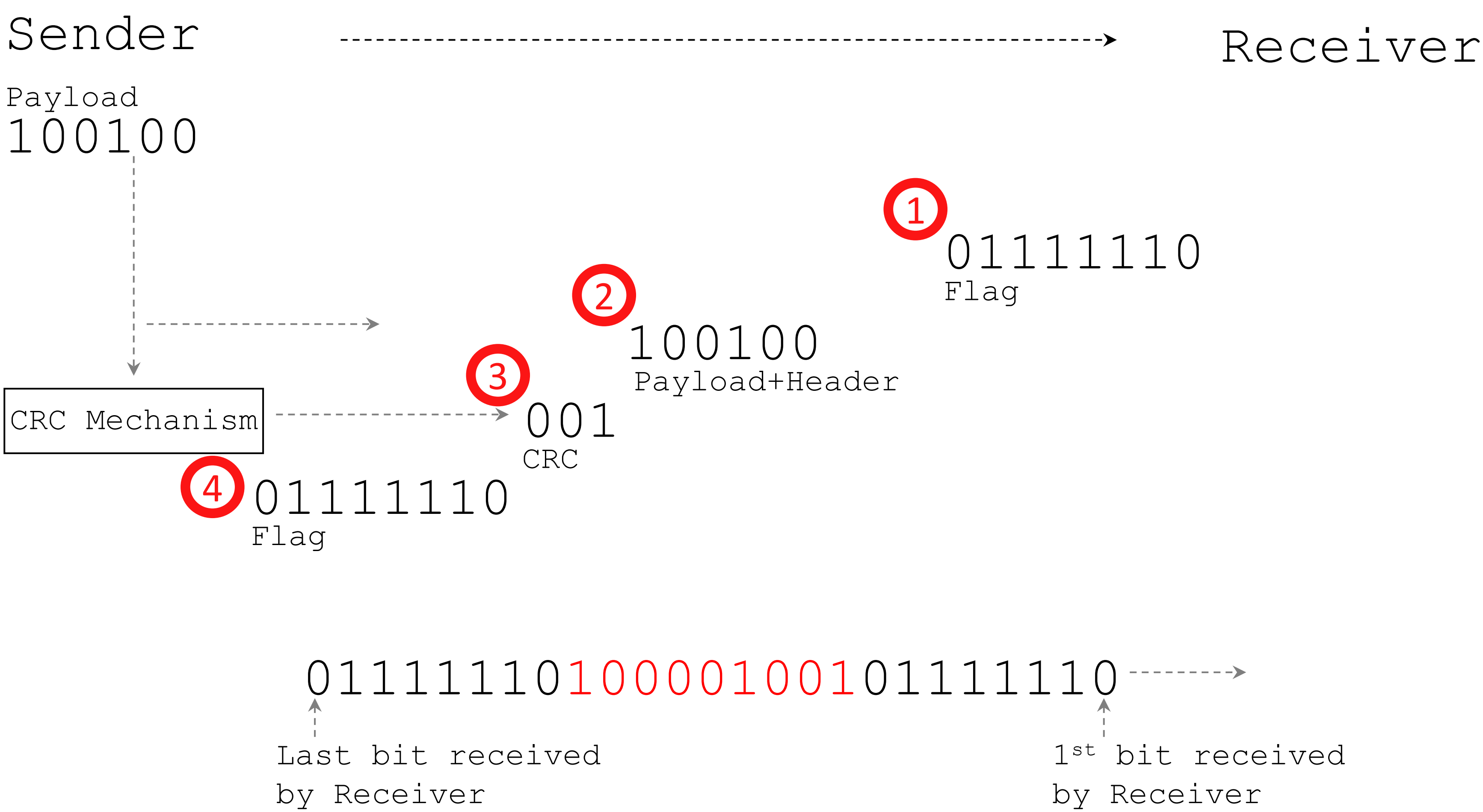
# CRC Calculation



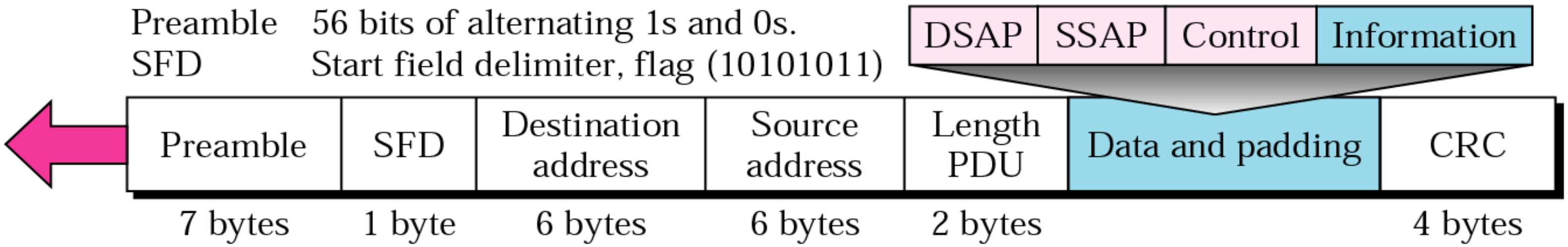
# CRC Calculation



# CRC Calculation



# 802.3 MAC Format




- 64-bit frame preamble (10101010) used to synchronize reception
  - 7 bit preamble (10101010) + 1 start flag (10101011)

- Maximum frame length: 1536 bytes

  ⇨ max 1500 bytes payload

- Minimum frame length: 64 bytes

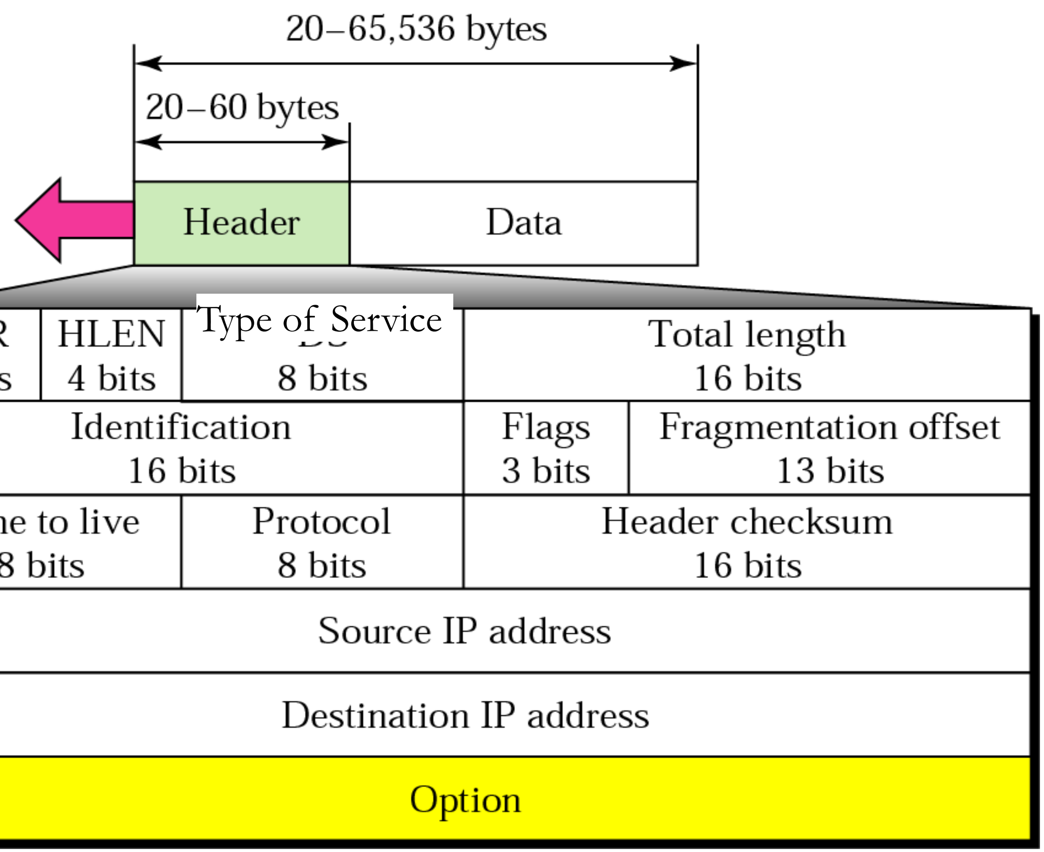
  ⇨ min 46 bytes payload

* Figure is courtesy of B. Forouzan

# Example from a Linux box

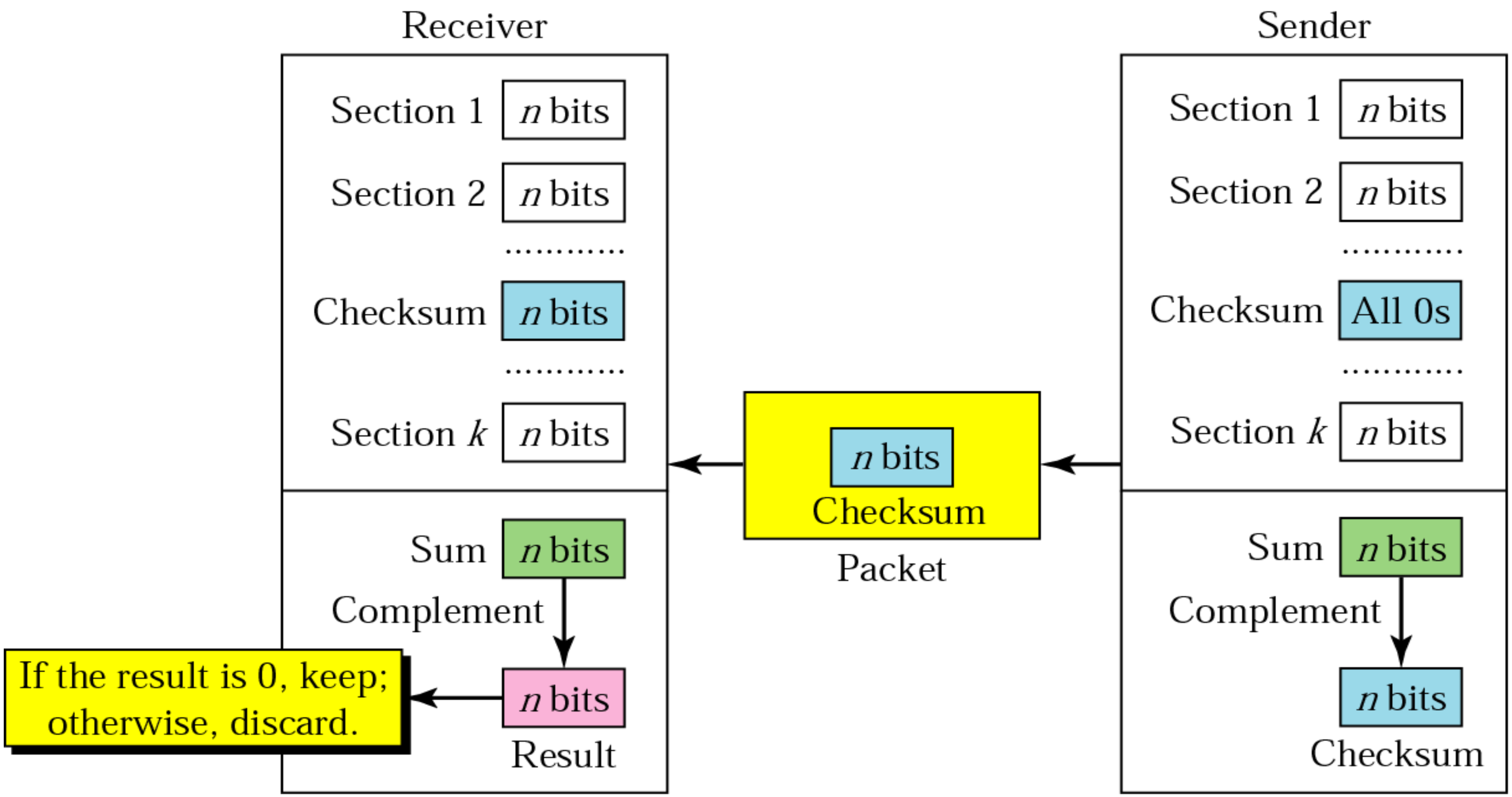
```
wlan0     Link encap:Ethernet  HWaddr 00:0b:81:89:56:ca
          inet addr:192.168.192.12  Bcast:192.168.192.255  Mask:255.255.255.0
          inet6 addr: fe80::20b:81ff:fe89:56ca/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:292 errors:0 dropped:374 overruns:0 frame:0
          TX packets:199 errors:0 dropped:2 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:47787 (46.6 KiB)  TX bytes:26749 (26.1 KiB)
```

# IP Datagram: Example for Checksum




* Figure is courtesy of B. Forouzan

# Checksum




* Figure is courtesy of B. Forouzan

# Checksum II

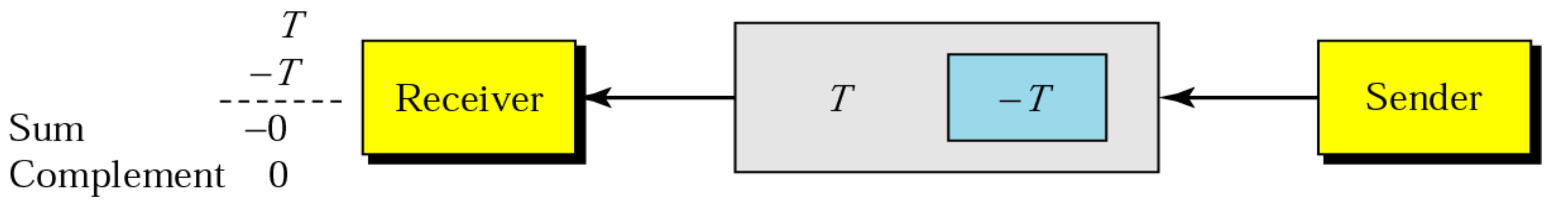
## Sender:

The unit is divided into k sections, each of n bits.

All sections are added using one's complement to get the sum.

The sum is complemented and becomes the checksum.

The checksum is sent with the data.




## Receiver:

The unit is divided into k sections, each of n bits.

All sections are added using one's complement to get the sum.

The sum is complemented.

If the result is zero, the data are accepted: otherwise, rejected.

* Figure is courtesy of B. Forouzan

# Example: Checksum

Sender:

10101001

00111001

------------

Sum 11100010

Checksum **00011101**

The data that is send:

10101001 00111001 **00011101**

# Example: Checksum

Sender:

| | |
|---|---|
| | 10101001 |
| | 00111001 |
| | ------------ |
| Sum | 11100010 |
| Checksum | **00011101** |

The data that is send:

10101001 00111001 **00011101**

Receiver:

| | |
|---|---|
| | 10101001 |
| | 00111001 |
| | 00011101 |
| Sum | 11111111 |
| Complement | **00000000** |

Complement: **00000000**

means that the frame is OK.

# Example: Checksum

Sender:

10101001

00111001

------------

Sum 11100010

Checksum **00011101**

The data that is send:

10101001 00111001 **00011101**

Receiver:

10101111

11111001

00011101

Partial Sum **1** 11000101

Carry **1**

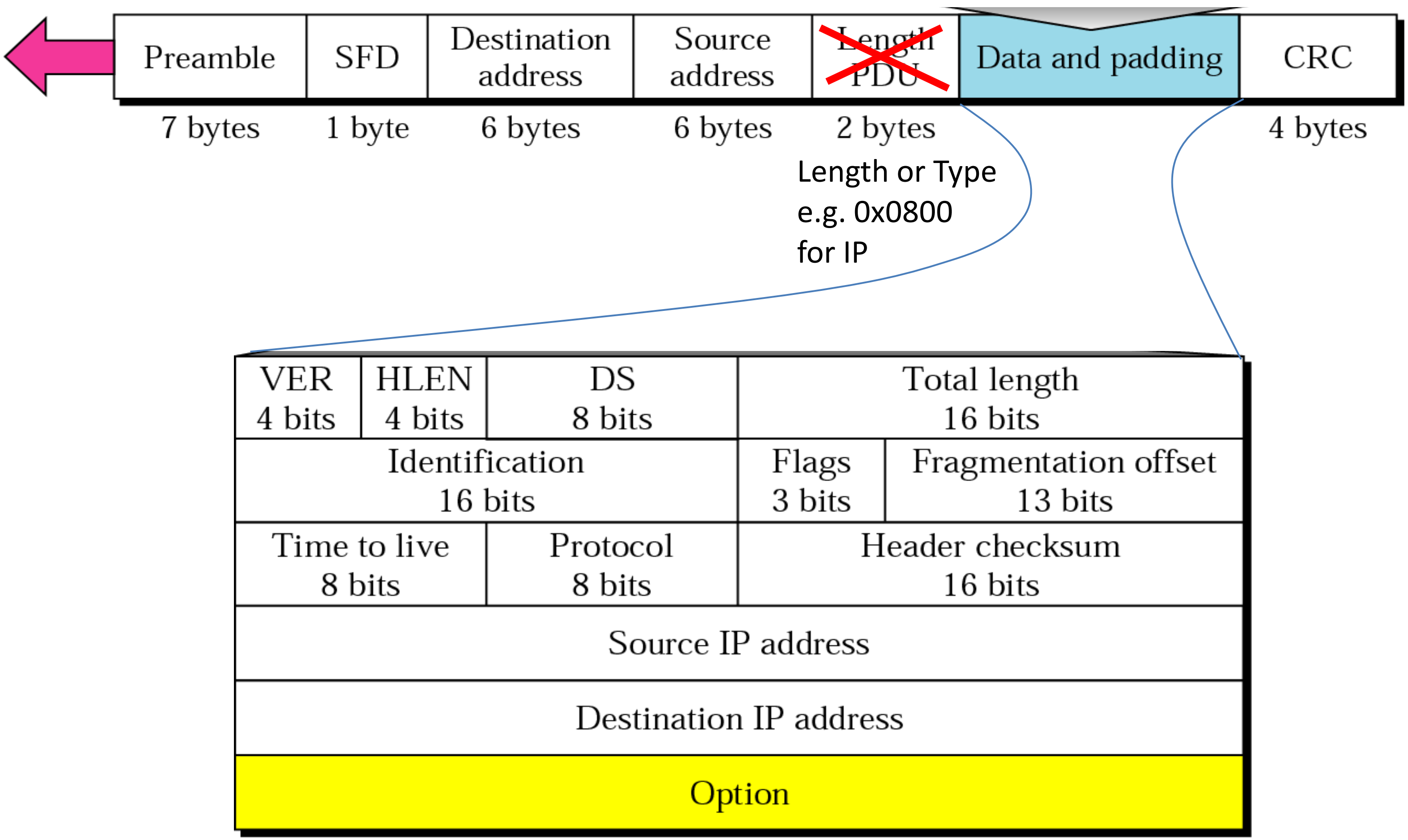
Sum 11000110

Complement **00111001**

Complement: **00111001**

means that the frame is corrupted.

# Ethernet & IP




* Figure is courtesy of B. Forouzan

# Sample TCP / IP Packet

```
0000  00 07 e9 7c 22 fc 00 11 93 85 e0 c4 08 00 45 00   ...|"........E.
0010  00 2c db 26 40 00 3f 06 0e 77 86 e2 20 37 86 e2   .,.&@.?..w.. 7..
0020  24 33 01 bd 12 3f 3d fa 0f b6 a8 6f 87 c0 50 18   $3...?=....o..P.
0030  bc 40 8a 7c 00 00 85 00 00 00 00 00               .@.|........
```

Ethernet Header:
src addr: 00 07 e9 7c 22 fc
dest addr: 00 11 93 85 e0 c4

IP Header:
src addr: 134.226.36.55
dest addr: 134.226.36.51

TCP Header:
src port: 445
dest port: 4671

NetBios Information

IP Header Checksum

The IP header is generally 20 byte and can be divided into units of 2 bytes/16 bits to calculate the checksum

# Summary: Detection of Errors

- Parity Check
- Cyclic Redundancy Check (CRC)
- Checksum

# Correction of Errors

- Error Correction through Retransmission
  - Parity, CRC, Checksum determine validity
  - If not valid, discard and wait for sender to retransmit

- Forward Error Correction
  - Determine the corrupted bit or bits at the receiver

# Hamming Code

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| d | d | d | $r_8$ | d | d | d | $r_4$ | d | $r_2$ | $r_1$ |

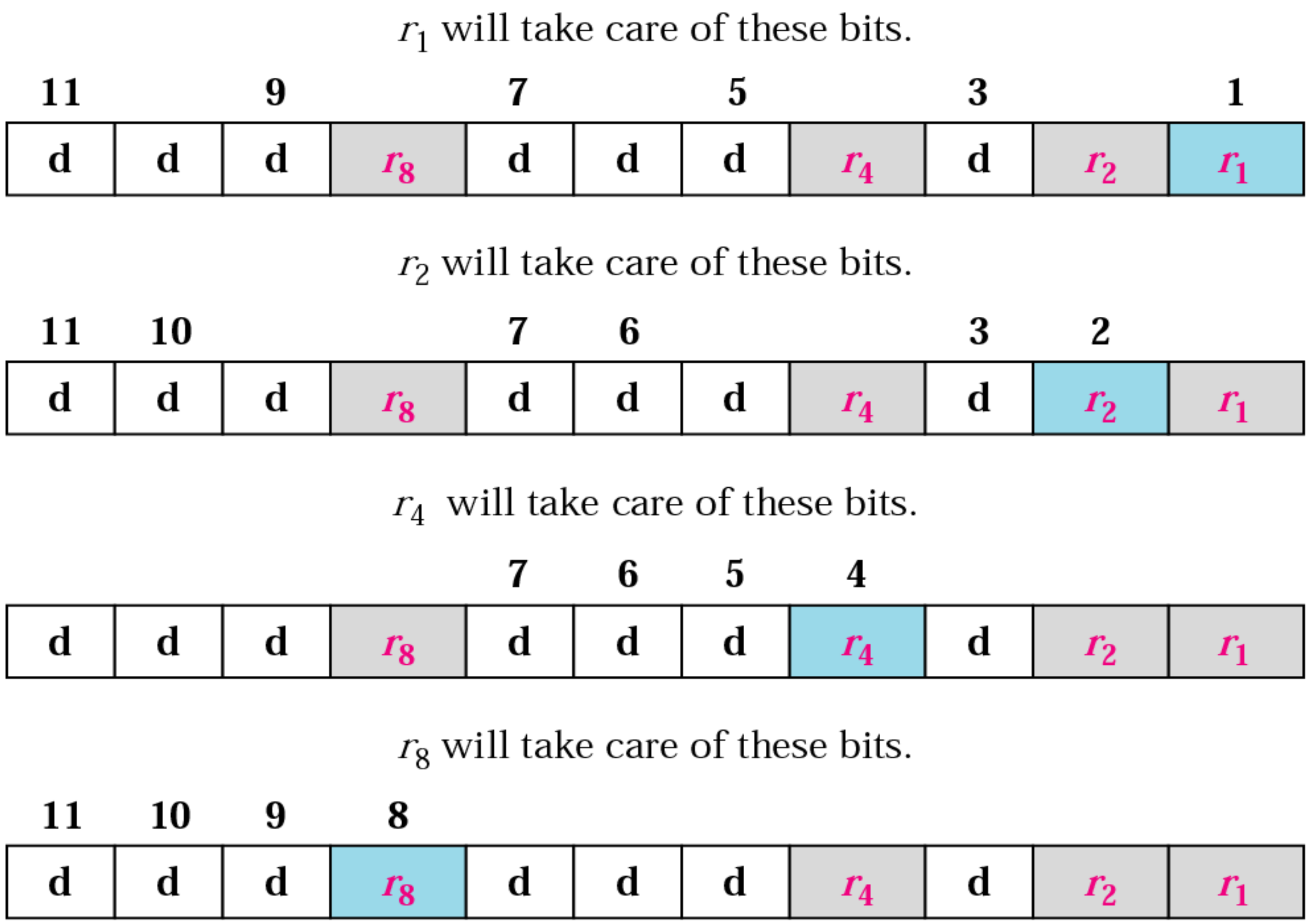
- Redundancy bits distributed throughout data bits
- Individual redundancy bits work as parity bits for specific data bits
  - e.g. $r_1$ is the parity bit for all odd numbers

    3 = binary 0011 7= binary 0111

    5 = binary 0101 9= binary 1001

* Figure is courtesy of B. Forouzan

# Redundancy Bits Calculation




* Figure is courtesy of B. Forouzan

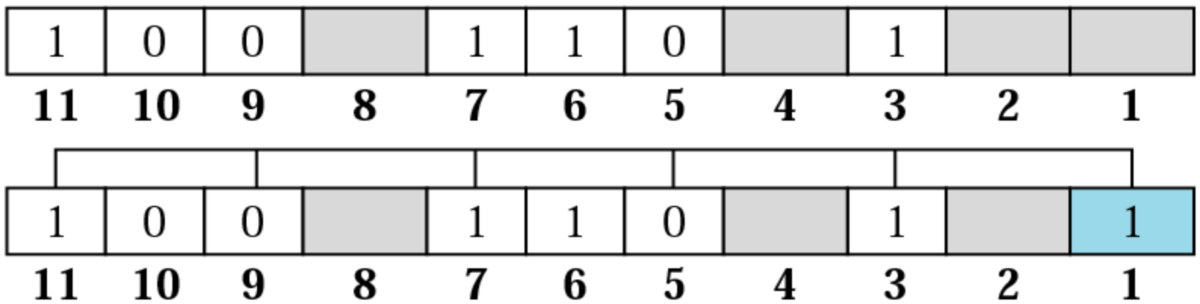
# Redundancy Bit Calculation

| 1 | 0 | 0 | | 1 | 1 | 0 | | 1 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** |

**Data:**
**1 0 0 1 1 0 1**

* Figure is courtesy of B. Forouzan

# Redundancy Bit Calculation

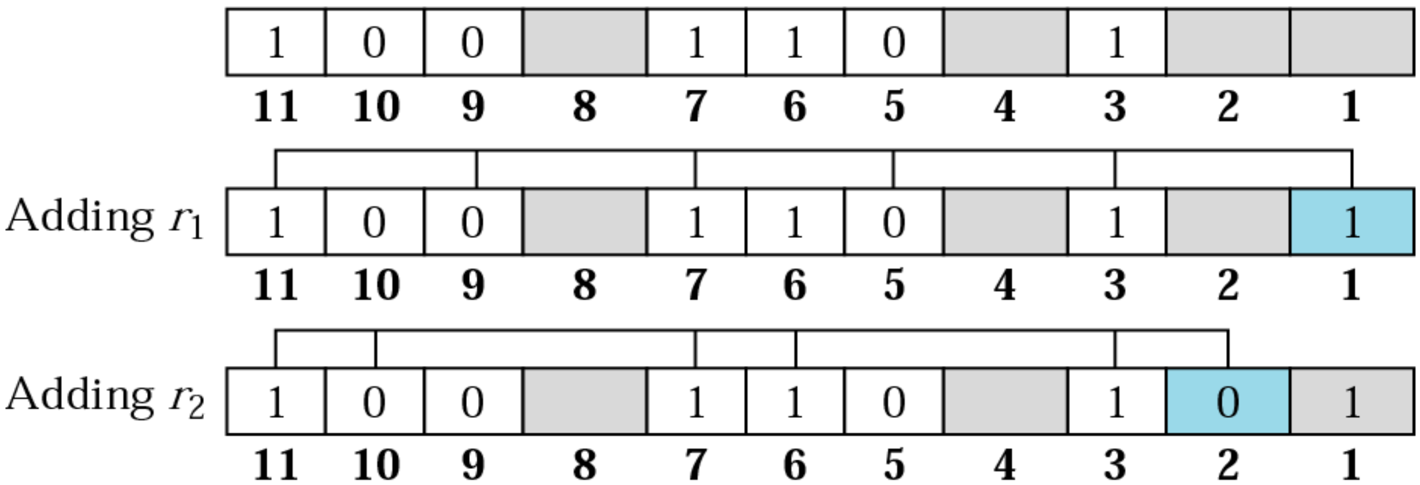



Adding $r_1$

**Data:**
**1 0 0 1 1 0 1**

* Figure is courtesy of B. Forouzan

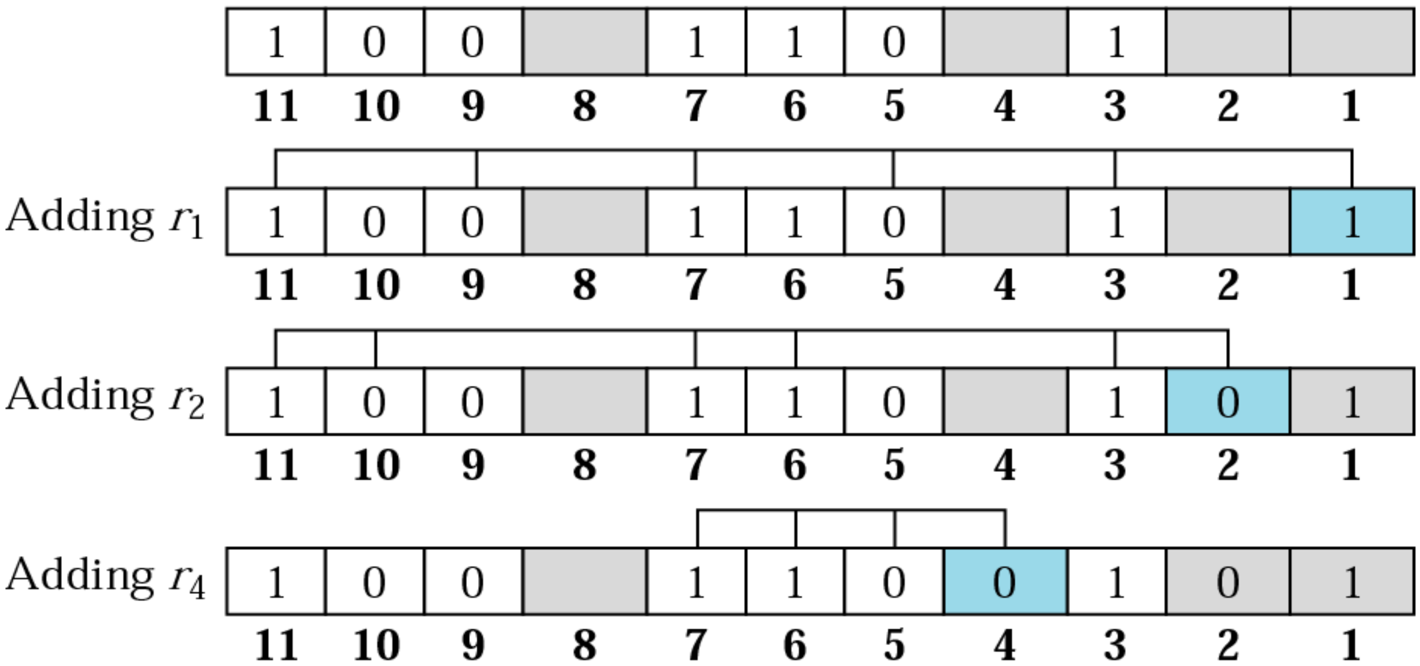
# Redundancy Bit Calculation




Data:
1 0 0 1 1 0 1

* Figure is courtesy of B. Forouzan

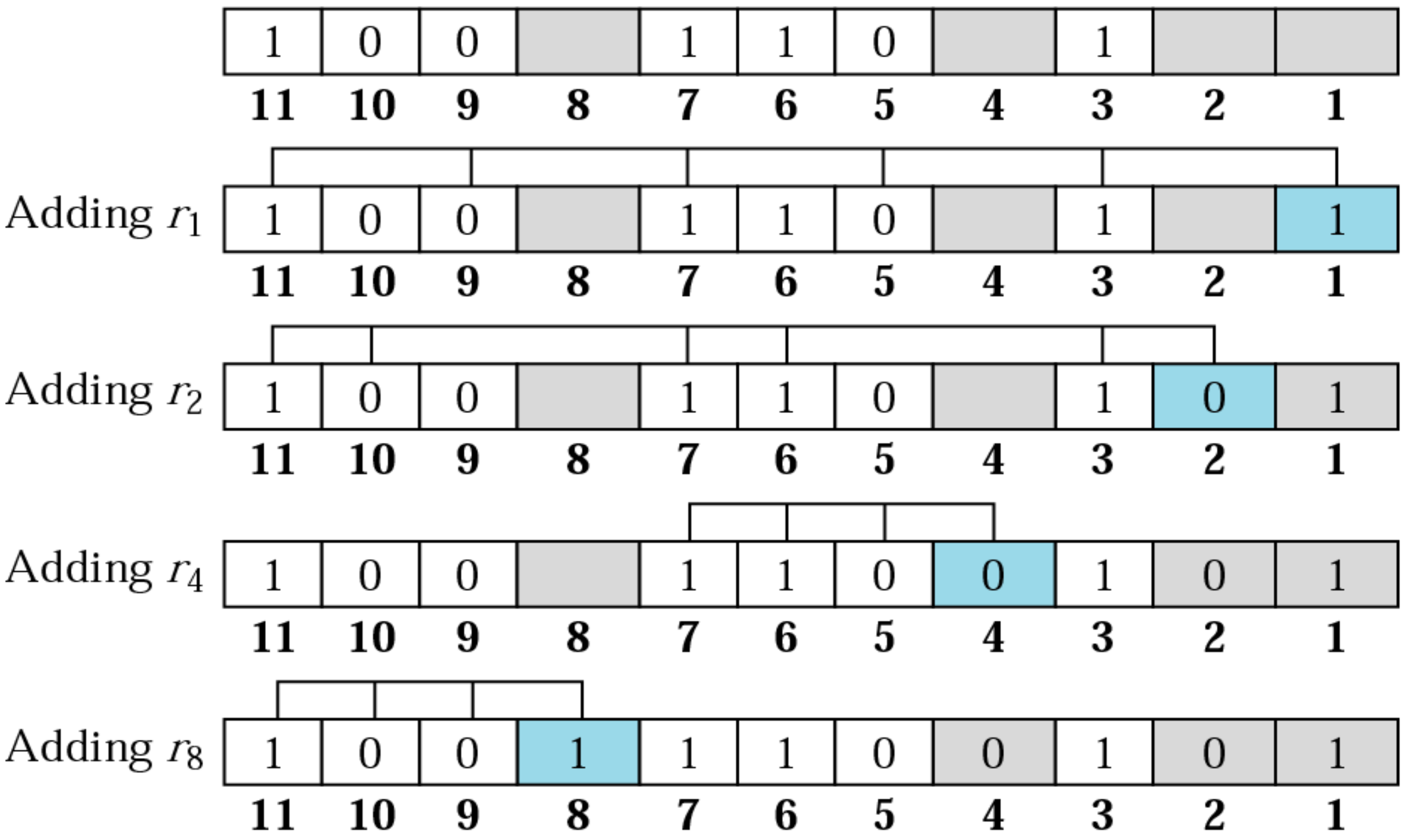
# Redundancy Bit Calculation




**Data:**
**1 0 0 1 1 0 1**

* Figure is courtesy of B. Forouzan

# Redundancy Bit Calculation

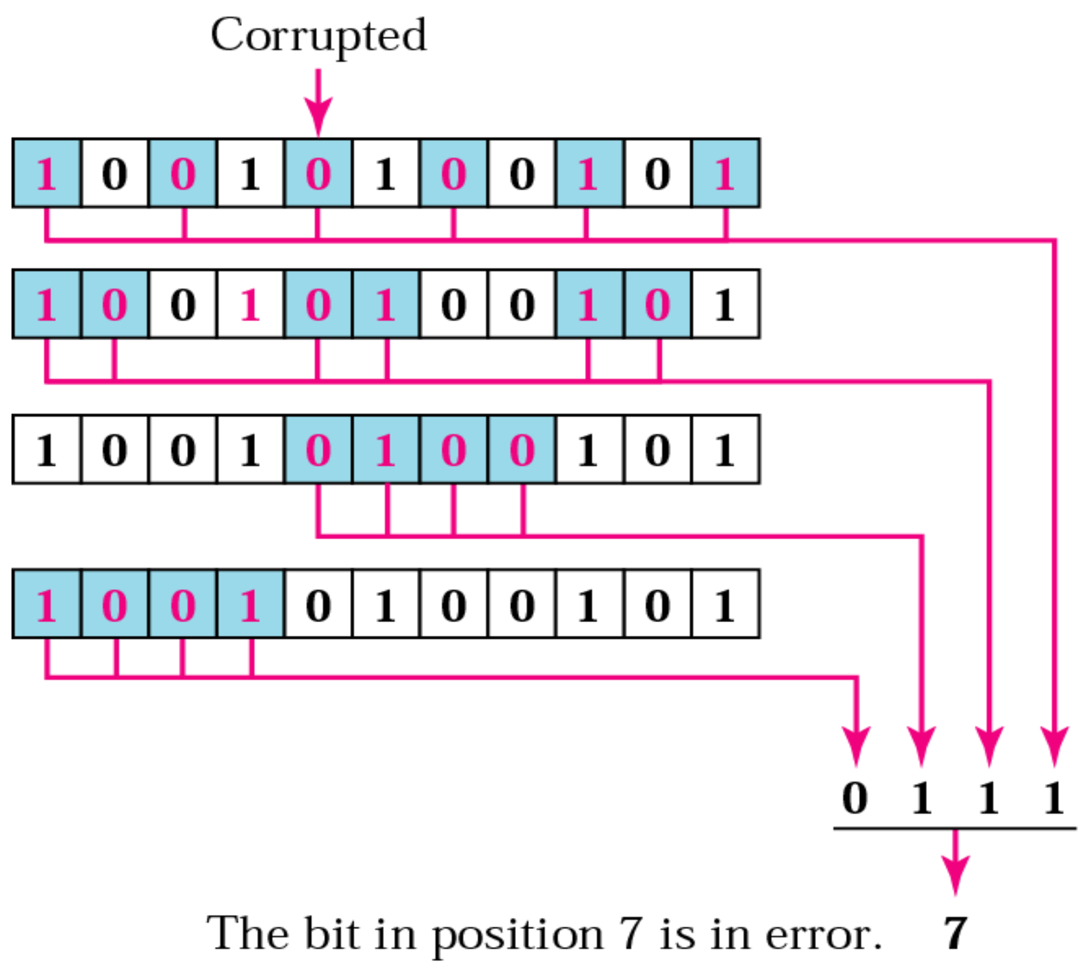



**Data:**
**1 0 0 1 1 0 1**

**Code:**
**1 0 0 1 1 1 0 0 1 0 1**

* Figure is courtesy of B. Forouzan

# Error Detection using Hamming Code




* Figure is courtesy of B. Forouzan

# Data and Redundancy Bits

| Number of data bits m | Number of redundancy bits r | Total bits m + r |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 5 |
| 3 | 3 | 6 |
| 4 | 3 | 7 |
| 5 | 4 | 9 |
| 6 | 4 | 10 |
| 7 | 4 | 11 |

* Figure is courtesy of B. Forouzan

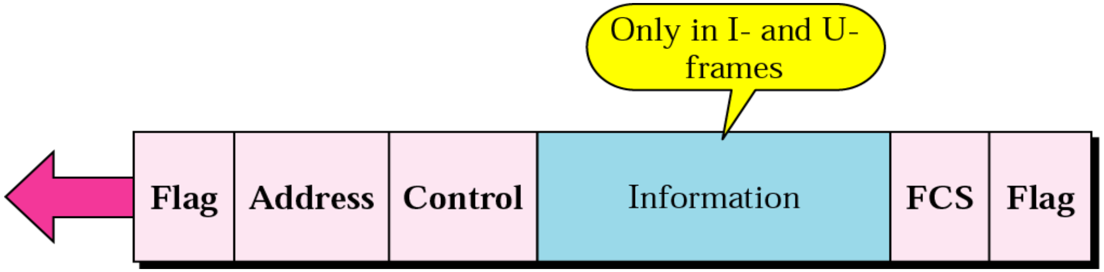
# Summary

- Types of Errors
  - Single-Bit & Burst Errors

- Detection of Errors
  - Parity Check / 2D Parity Check
  - CRC ← Sequence of bits
  - Checksum ← Chunks of bits

- Correction of Errors
  - Error Correction by Retransmission
  - Forward Error Correction – Hamming Code

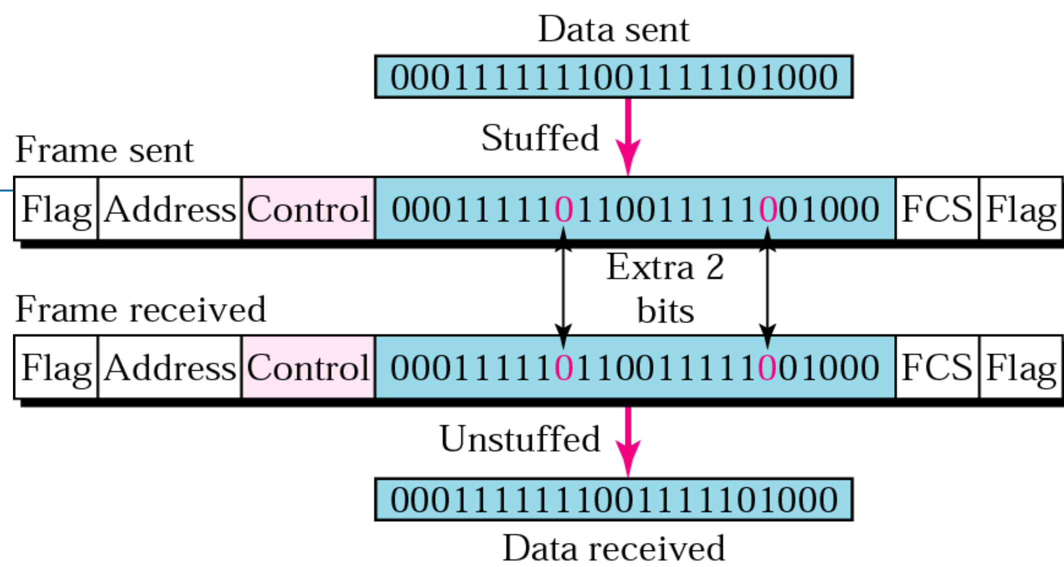
# HDLC frame




- Flag= 01111110
  - specifies beginning and end of frame
- Address
  - specifies secondary station
  - as either sender or receiver
- Control
  - specifies type of frame and seq.&ack. number
- Frame Check Sequence (FCS)
  - either 16- or 32-bit CRC

* Figure is courtesy of B. Forouzan

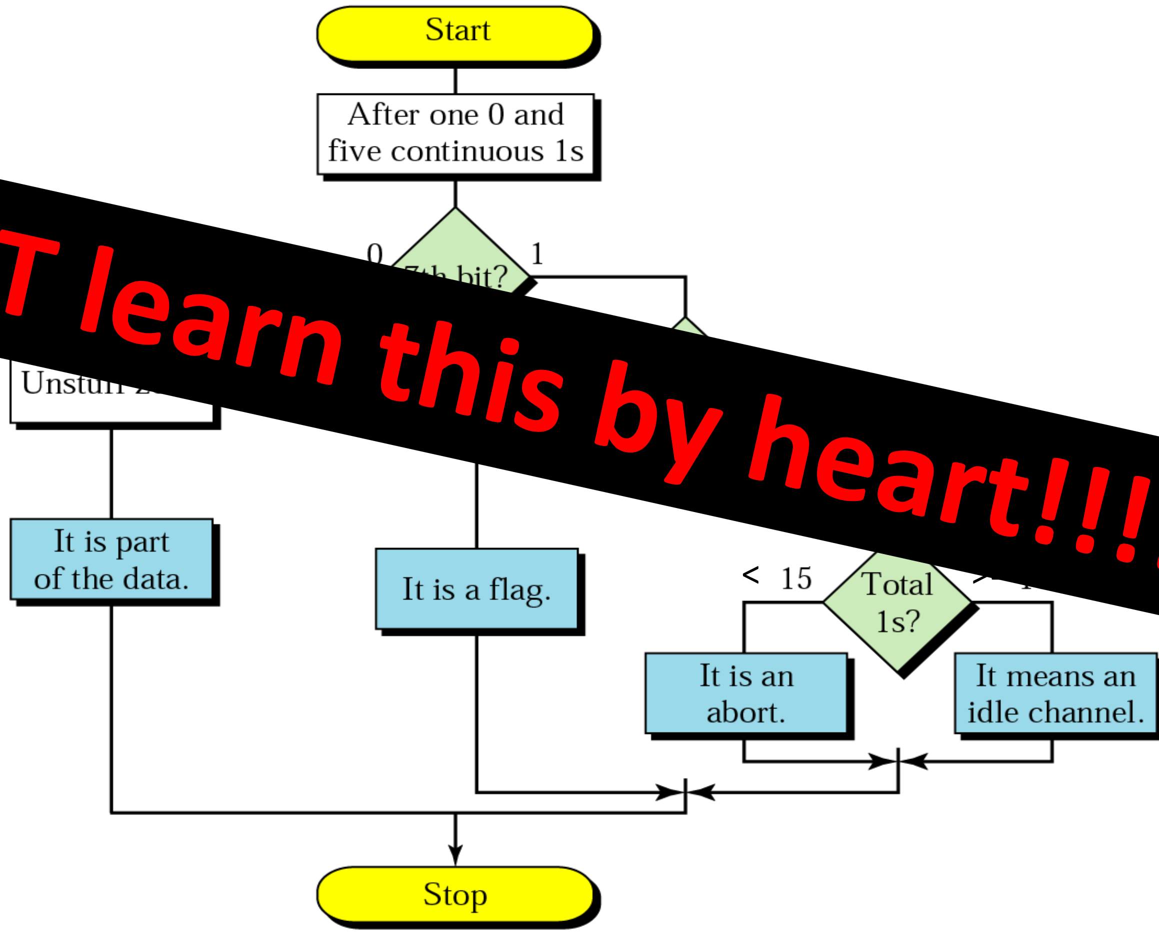
# Bit-Stuffing




- Bit stuffing used to avoid confusion with data containing same combination as flag 01111110
  - 0 inserted after every sequence of five 1s
  - If receiver detects five 1s
    - it checks next bit
    - If 0, it is deleted
    - If 1 and seventh bit is 0, accept as flag
    - If sixth and seventh bits 1, sender is indicating abort

* Figure is courtesy of B. Forouzan

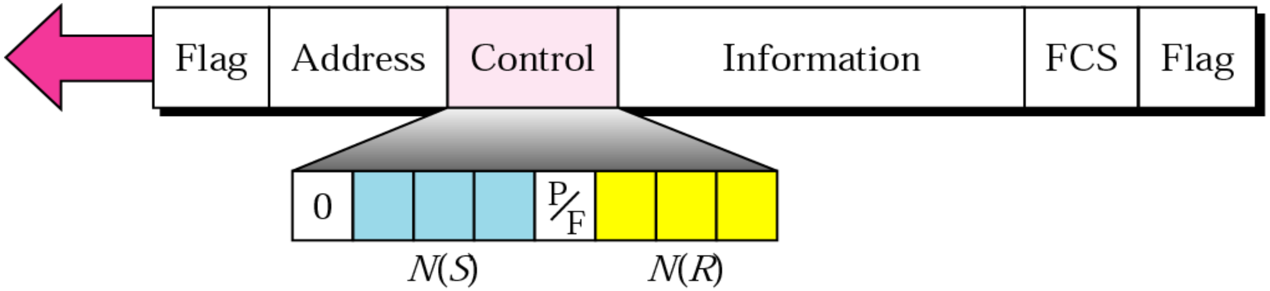
# Bit stuffing in HDLC




**Do NOT learn this by heart!!!!!**

* Figure is courtesy of B. Forouzan

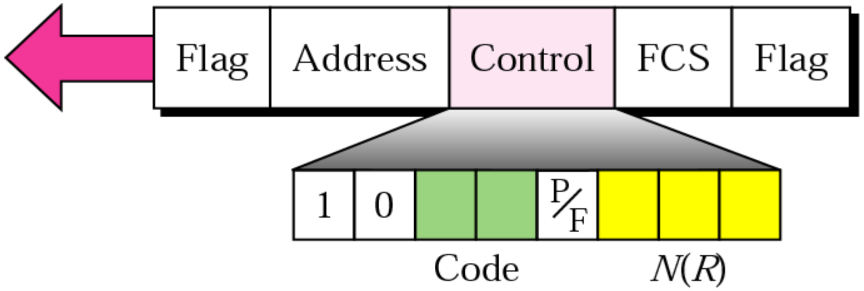
# I-Frame




- N(S)
  - Sequence Number of Sender
- N(R)
  - Sequence Number of Receiver
- P/F
  - Poll/Final bit
  - Set by Primary station as request for information
  - Set by Secondary station to signal response or to signal final frame of a transmission

* Figure is courtesy of B. Forouzan

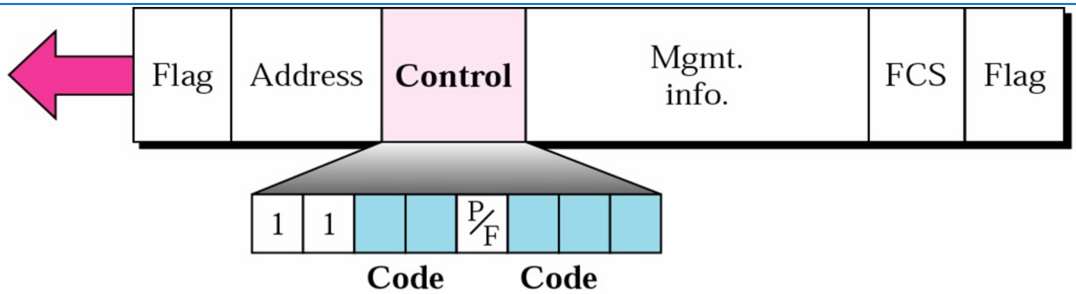
# S-Frame Control Field




- Code 00 = Receive Ready (RR)
  - Acknowledge frames & waiting for more
- Code 10 = Receive Not Ready (RNR)
  - Acknowledge frames & busy right now
- Code 01 = Reject (REJ)
  - Go-Back-N NAK
- Code 11 = Selective Reject (SREJ)
  - Selective Repeat NAK

* Figure is courtesy of B. Forouzan

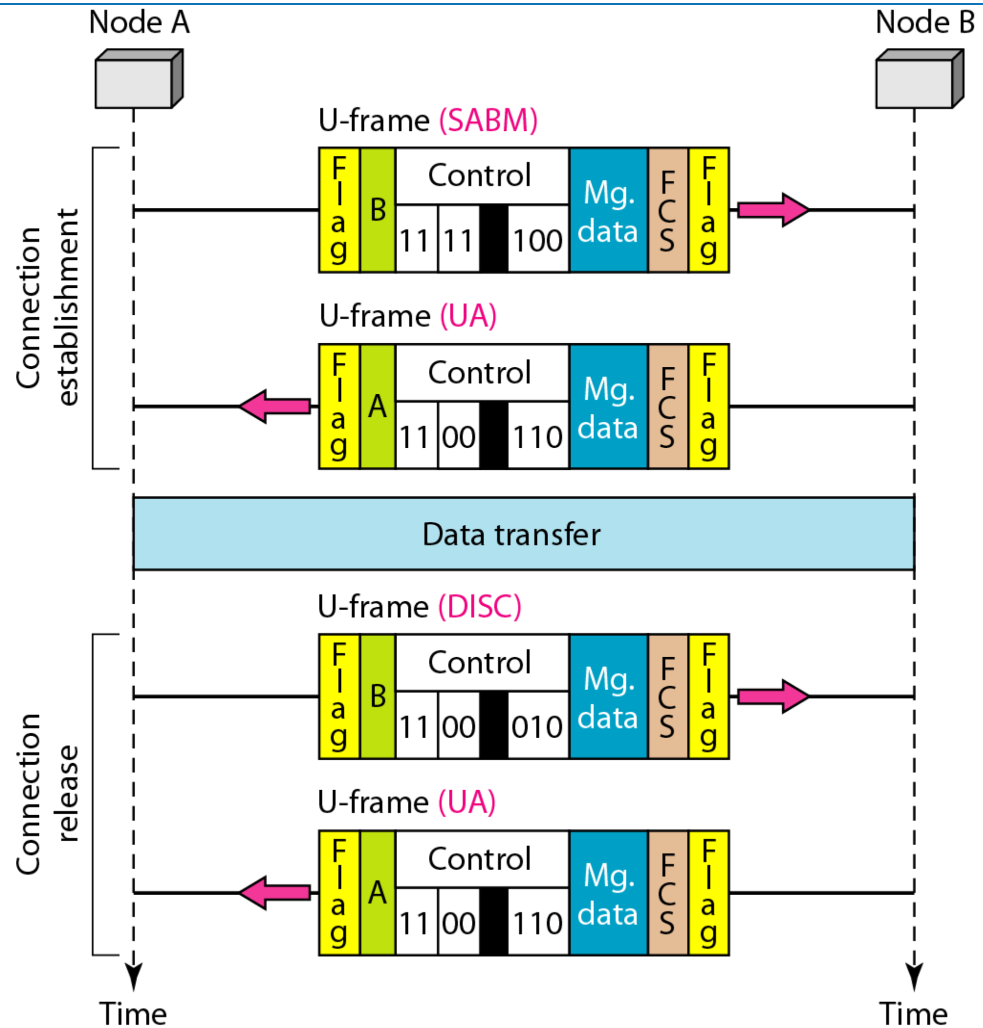
# U-Frame Control Field




| Code | Command/Response | Meaning |
|---|---|---|
| 00 001 | SNRM | Set normal response mode |
| 11 100 | SABM | Set asynchronous balanced mode |
| 00 100 | UP | Unnumbered poll |
| 00 000 | UI | Unnumbered information |
| 00 110 | UA | Unnumbered acknowledgment |
| 00 010 | DISC | Disconnect |
| 10 000 | SIM | Set initialization mode |
| 11 001 | RSET | Reset |
| 11 101 | XID | Exchange ID |
| 10 001 | FRMR | Frame reject |

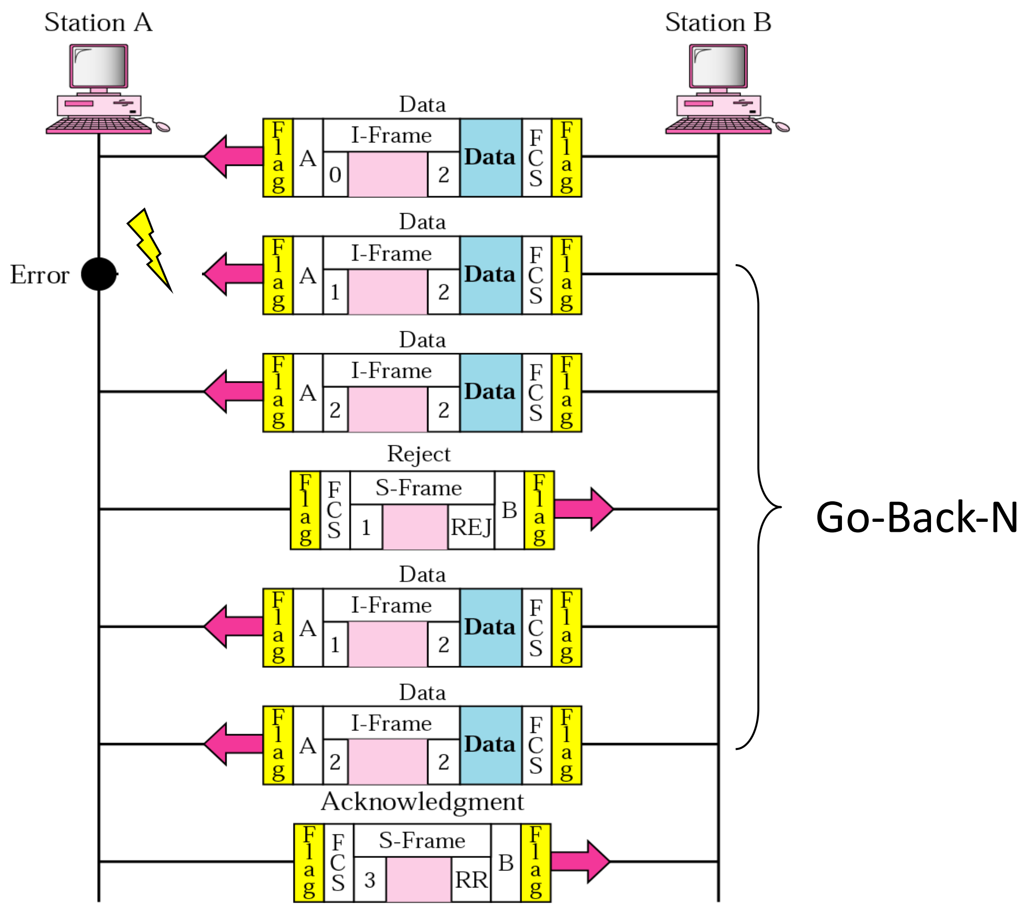
* Figure is courtesy of B. Forouzan

# Connection & Disconnection




* Figure is courtesy of B. Forouzan

# Piggybacking with Error




* Figure is courtesy of B. Forouzan

# HDLC – Why?

- ‘should give you a feeling for a protocol

- It includes most of the basic mechanisms
  - Framing
  - Addressing
  - Bit-stuffing
  - Flow/Error control

- Once you can run through HDLC in your head, you understand the basics of link layer protocols

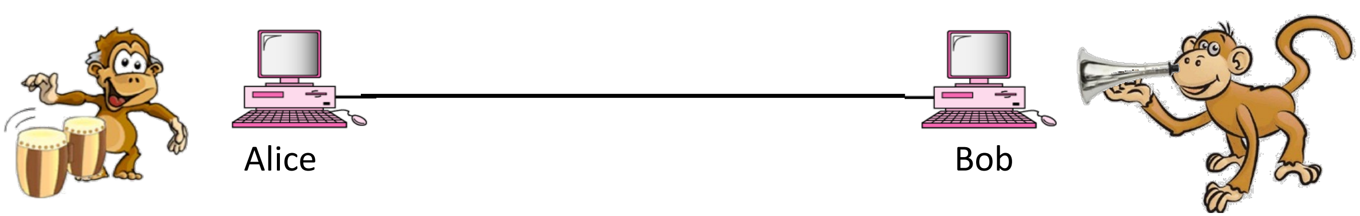
# Binary Example

01111110011111000111101101010110101111110

0111111001111010110001101110110101111110

01111110011111000110000101011110101111110

0111111001111010110001101110110101111110



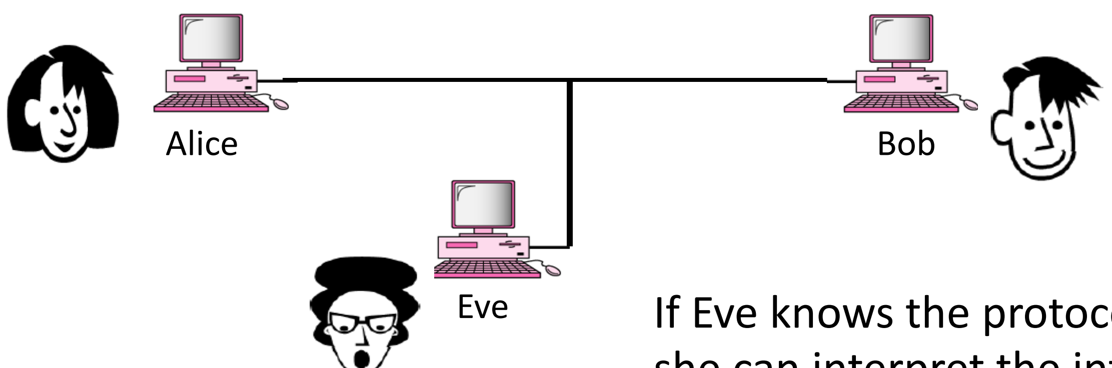
# Binary Example

01111110011111000111101101010110101111110

0111111001111010110001101110110101111110

01111110011111000110000101011110101111110

0111111001111010110001101110110101111110




If Eve knows the protocol,
she can interpret the information

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin