

Unit-2

Data Link Layer

Errors

Problem: *All* data communications systems are susceptible to errors

- The physics of the universe causes errors.
- Devices fail and/or equipment does not meet engineering standards.

Thus: We need ways/mechanisms to control and recover from errors.

Note: Small errors are often more difficult to detect than complete failures.

Three Main Sources of Transmission Errors

1. *Interference*

Example: Electromagnetic radiation interferes with electrical signals

2. *Distortion*

Rule: All physical systems distort signals

3. *Attenuation*

Example: As a signal passes across a medium, the signal becomes weaker

Reducing Errors

Shannon's Theorem suggests one way to reduce errors:

Increase the signal-to-noise ratio (either by increasing the signal or lowering noise if possible)

Unfortunately, it is not always possible to change the signal-to-noise ratio.

Example: Mechanisms like shielded wiring can help lower noise but a physical transmission system is always susceptible to some noise

Bottom Line: Noise/interference cannot be eliminated completely

Fortunately, many transmission errors can be detected.

In some cases, errors can even be corrected automatically (but hardly ever!)

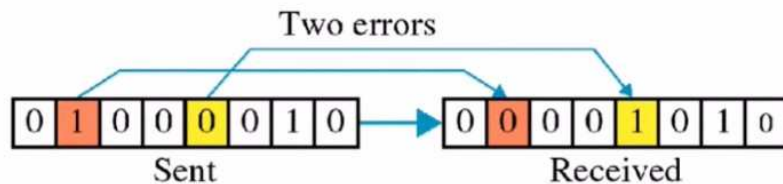
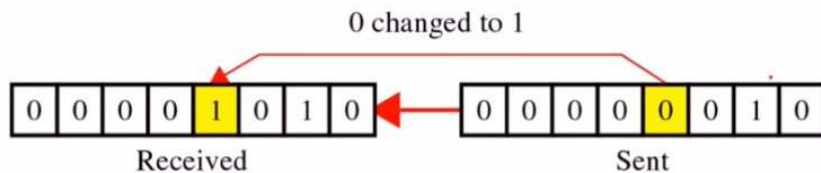
Conclusion: Error handling is a tradeoff between the *need for detecting errors* and the *additional overhead for error detection and/or correction*.

Errors

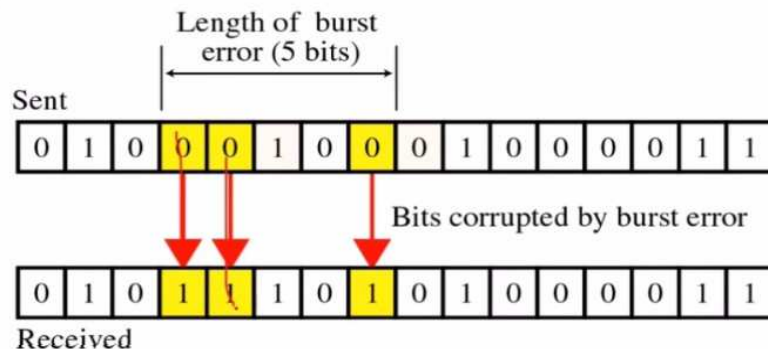
When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits leads to spurious data being received by the destination and are called errors.

Effect of Transmission Errors on Data

Type Of Error	Description
Single Bit Error	A single bit in a block of bits is changed and all other bits in the block are unchanged (often results from very short-duration interference)
Burst Error	Multiple bits in a block of bits are changed (often results from longer-duration interference)
Erasure (Ambiguity)	The signal that arrives at a receiver is ambiguous (does not clearly correspond to either a logical 1 or a logical 0 (can result from distortion or interference)



~ means that 2 or more consecutive bits in the data unit have changed



Two key Characteristics of errors:

1. **BER bit error rate:** Probability P of a single bit being corrupted in a defined time interval.
2. **Random (or burst) errors:** Whether the errors occur as random single-bit errors or as groups of contiguous errors.

Redundancy

Additional bits are used for correcting and detecting errors.

- Error Detection: Yes or No
- Error Correction: Exact no of bits corrupted & their location

Two things in case of Error Detection:

1) **Forward Error Correction:**

- Analyze the Data
- Redundant Bits helps we guess, what is corrupted
- Try to fix it

2) **Retransmission:**

- Corruption confirmed and asks sender to resend data

Error Control

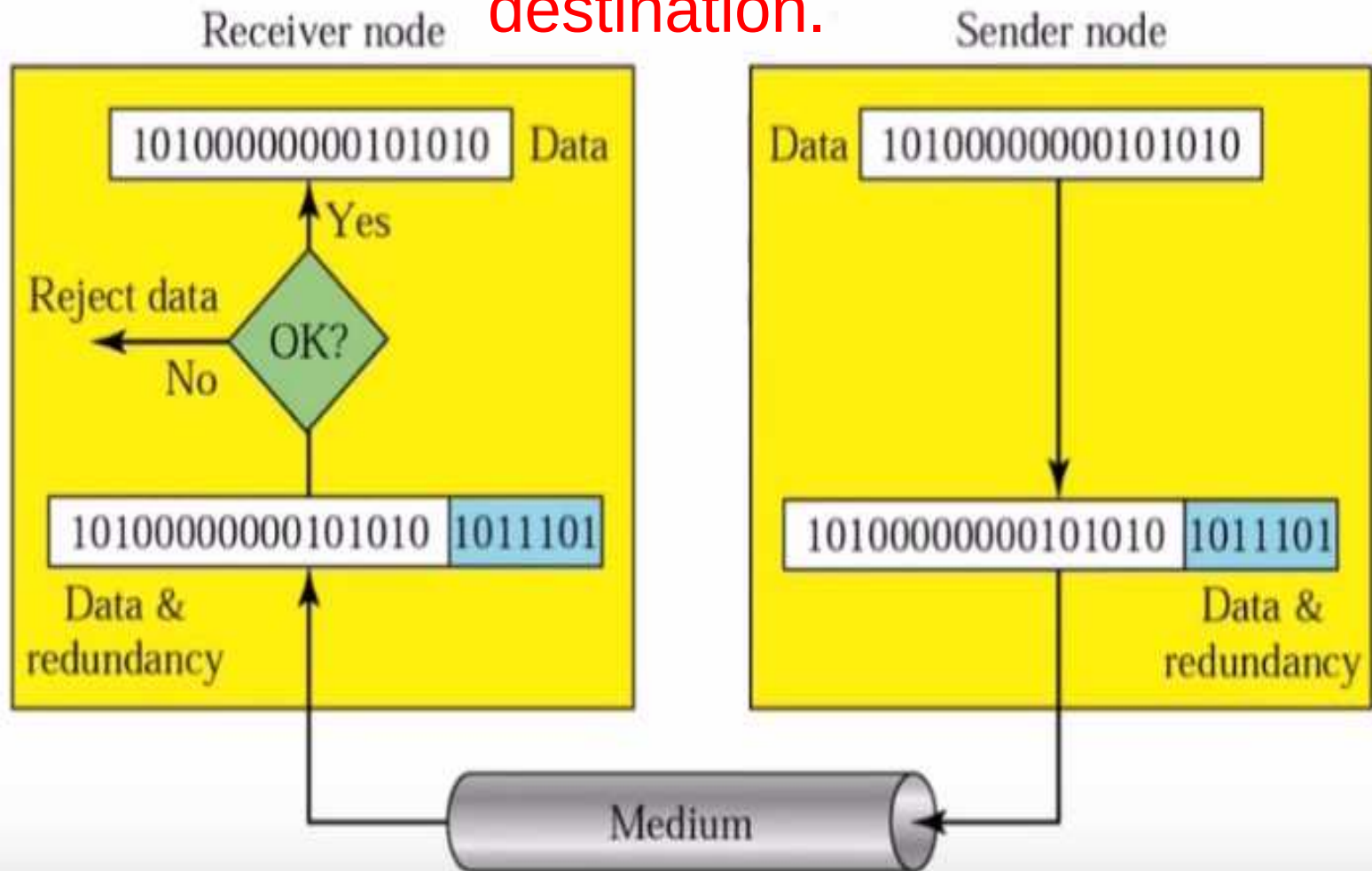
Error control can be done in two ways

- **Error detection** – Error detection involves checking whether any error has occurred or not. The number of error bits and the type of error does not matter.
- **Error correction** – Error correction involves ascertaining the exact number of bits that has been corrupted and the location of the corrupted bits.

For both error detection and error correction, the sender needs to send some additional bits along with the data bits. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it removes the redundant bits before passing the message to the upper layers.

Error Detection uses the concept of redundancy which means adding extra bits for detecting errors at destination.

❑ **Redundancy**



Error Detection Techniques

There are three main techniques for detecting errors in frames: Parity Check, Checksum and Cyclic Redundancy Check (CRC).

Parity Check

The parity check is done by adding an extra bit, called parity bit to the data to make a number of 1s either even in case of even parity or odd in case of odd parity.

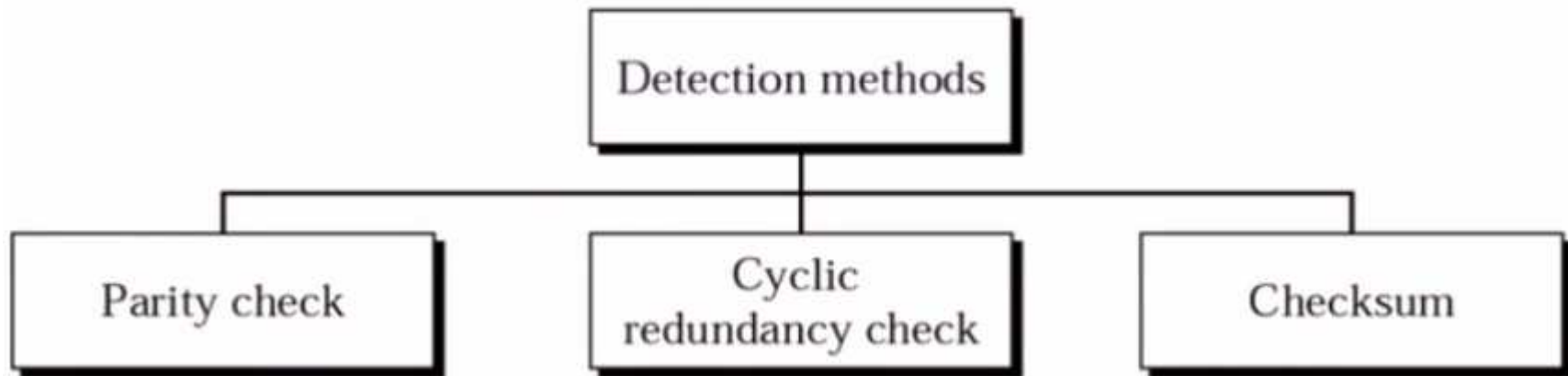
While creating a frame, the sender counts the number of 1s in it and adds the parity bit in the following way

- In case of even parity: If a number of 1s is even then parity bit value is 0. If the number of 1s is odd then parity bit value is 1.
- In case of odd parity: If a number of 1s is odd then parity bit value is 0. If a number of 1s is even then parity bit value is 1.

On receiving a frame, the receiver counts the number of 1s in it. In case of even parity check, if the count of 1s is even, the frame is accepted, otherwise, it is rejected. A similar rule is adopted for odd parity check.

The parity check is suitable for single bit error detection only.

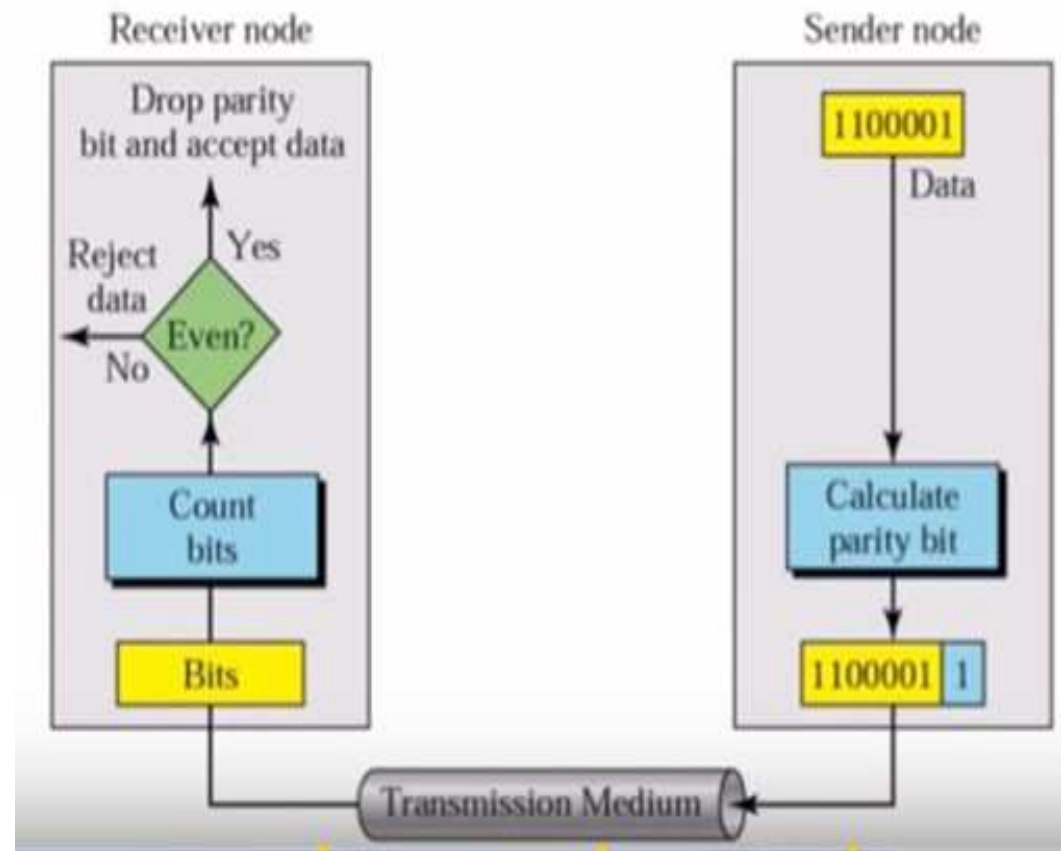
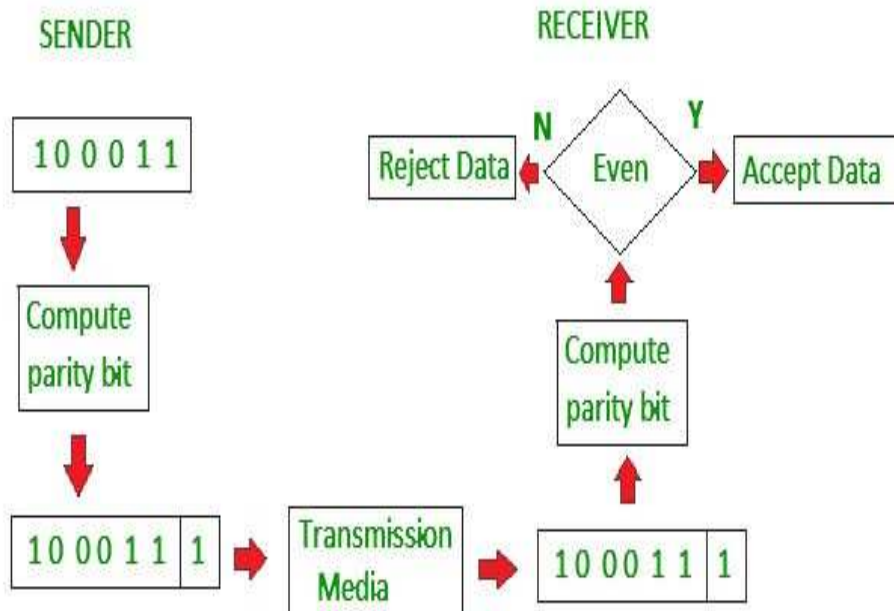
□ Detection methods



Parity Check

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

❖ Simple parity check



PARITY BITS

Type of bit parity	Successful transmission scenario
Even parity	<p>A wants to transmit: 1001 A computes parity bit value: $1+0+0+1 = 0$ A adds parity bit and sends: 10010 B receives: 10010 B computes parity: $1+0+0+1+0 = 0$ (Even) B reports correct transmission after observing expected even result.</p>
Odd parity	<p>A wants to transmit: 1001 A computes parity bit value: $1+0+0+1 = 1$ A adds parity bit and sends: 10011 B receives: 10011 B computes overall parity: $1+0+0+1+1 = 1$ (Odd) B reports correct transmission after observing expected odd result.</p>

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

11101110 11011110 11100100 11011000 11001001

receiver without being corrupted in transmission.

11101110 11011110 11100100 11011000 11001001

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

Parity bit examples

sequence of seven bits	with eighth even parity bit:	with eighth odd parity bit:
0100010	0100010 <u>0</u>	0100010 <u>1</u>
1000000	1000000 <u>1</u>	1000000 <u>0</u>

Error Detection and Correction

A variety of mathematical techniques have been developed that overcome errors during transmission and increase reliability.

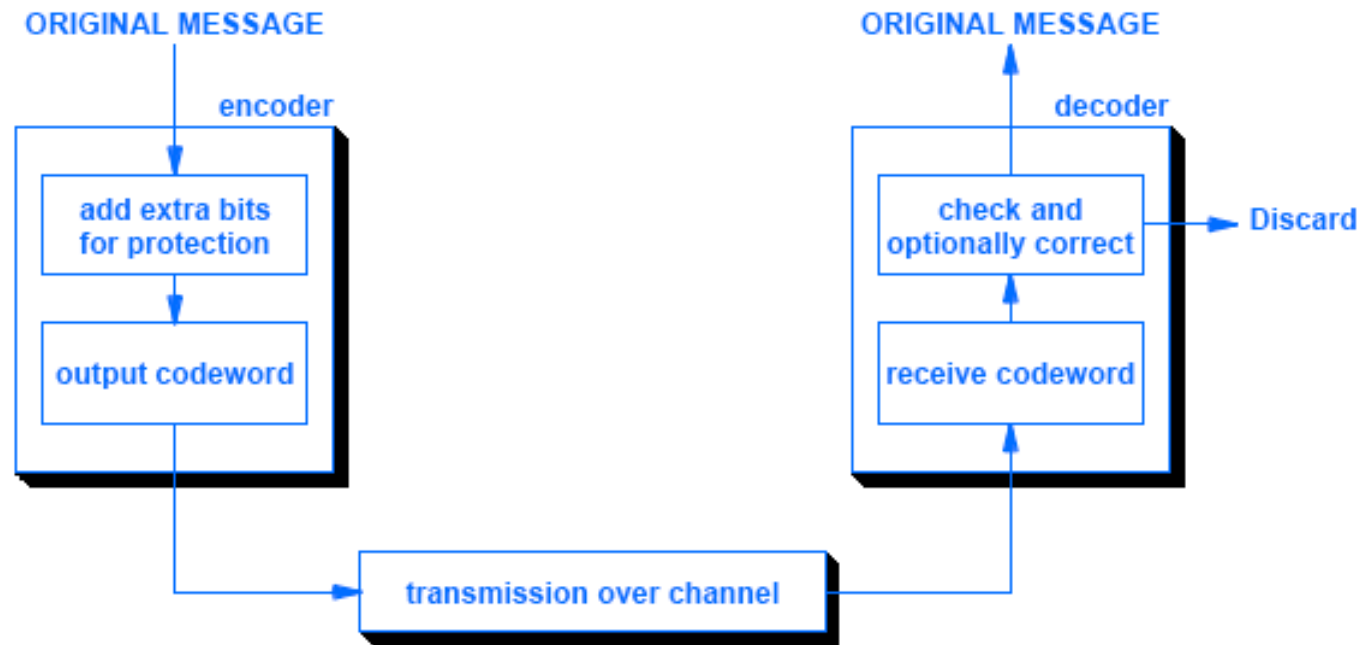
These techniques are known collectively as *channel coding*.

Two primary approaches:

1. **Error Control**
 - **Forward Error Control (FEC) mechanisms**
 - **Backward Error Control (BEC) mechanisms**
2. **Automatic Repeat reQuest (ARQ) mechanisms**

Error Control (FEC & BEC)

Basic idea of Error Control: Add additional information to the data that allows a receiver to verify that data arrives correctly and to correct errors (if possible).



Backward error control (error detecting): Each transmitted character or frame contains additional information so that the receivers can detect but not correct errors. A retransmission scheme must also be used.

Forward error control (error correcting): Each transmitted character or frame contains additional information so that the receivers can detect and correct errors.

Parity

Simplest method for detecting bit errors: *Single Parity Checking (SPC)*

Add the # of 1 bits in the code together (modulo 2) and choose the parity bit so that the total number of one bits is:

- even (even parity) or
- odd (odd parity)

A parity bit can be used to detect 1-bit errors in the code.

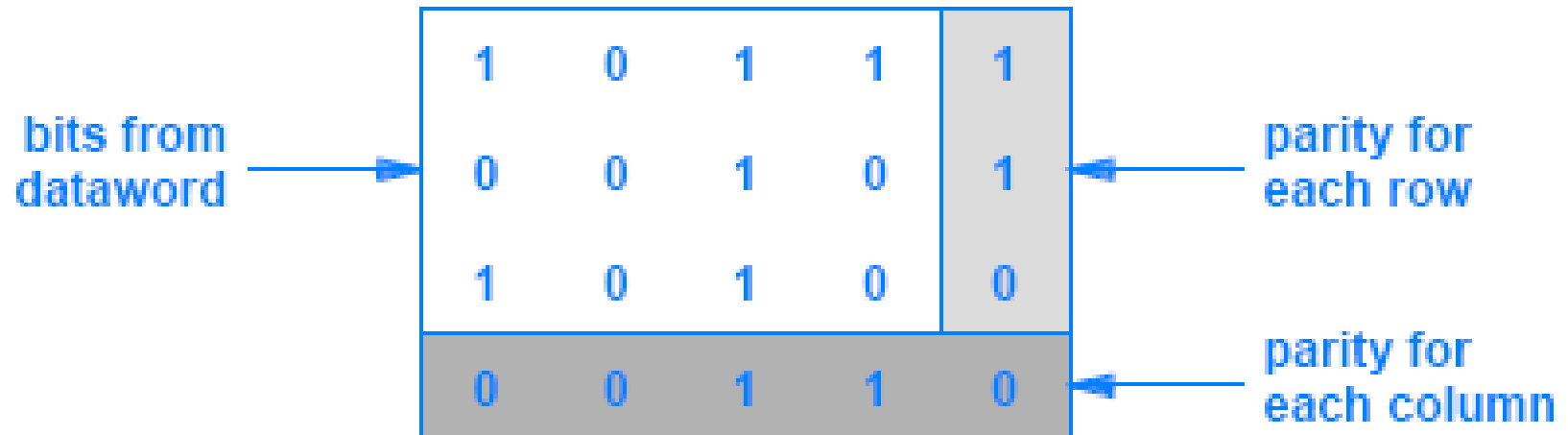
Original Data	Even Parity	Odd Parity
0 0 0 0 0 0 0	0	1
0 1 0 1 1 0 1 1	1	0
0 1 0 1 0 1 0 1	0	1
1 1 1 1 1 1 1 1	0	1
1 0 0 0 0 0 0 0	1	0
0 1 0 0 1 0 0 1	1	0

SPC is a weak form of channel coding that can detect errors but cannot correct errors.

An single-bit parity mechanism can only handle errors where an odd number of bits are changed

Two-Dimensional Parity (Block sum check)

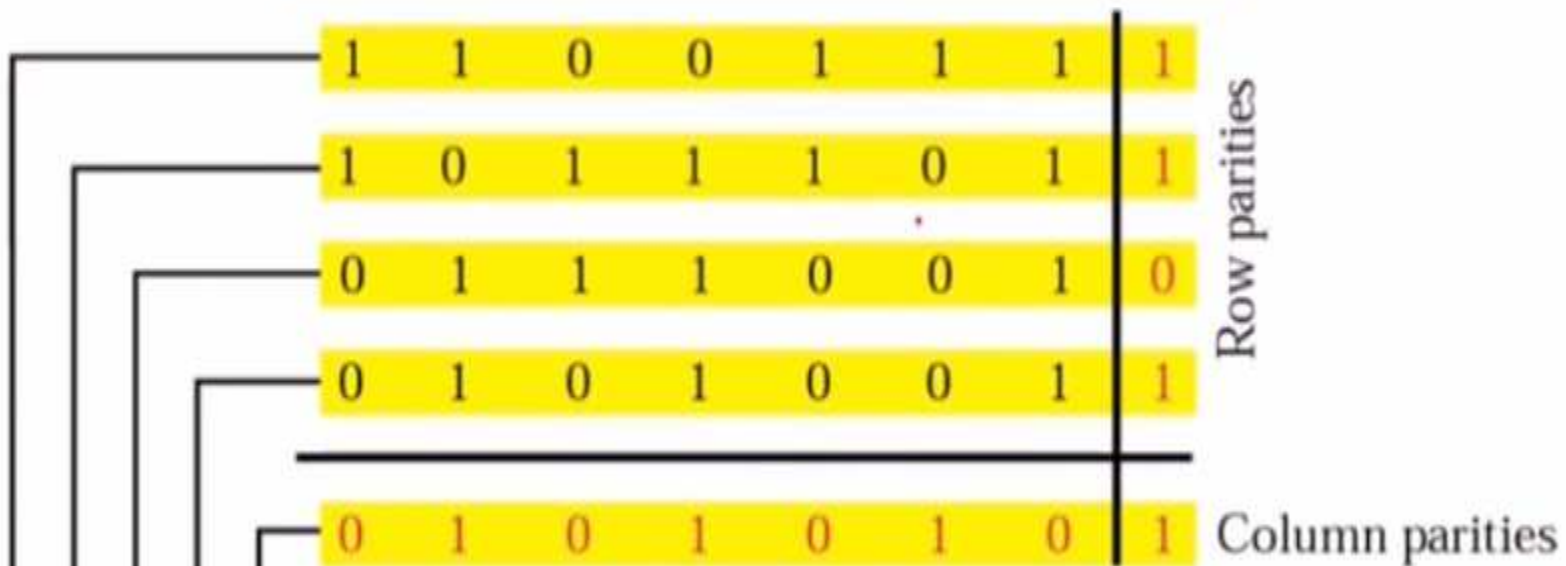
- Used with blocks of characters
- Use a **row parity** bit for each byte
- Use a **column parity** for each bit column position in the complete frame



Two-Dimensional Parity Check

Original data

1100111 1011101 0111001 0101001



11001111 10111011 01110010 01010011 01010101

Data and parity bits

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials.

- Here, the sender performs binary division of the data segment by the divisor. It then appends the remainder called CRC bits to the end of the data segment. This makes the resulting data unit exactly divisible by the divisor.
- The receiver divides the incoming data unit by the divisor. If there is no remainder, the data unit is assumed to be correct and is accepted. Otherwise, it is understood that the data is corrupted and is therefore rejected.

Example-2

Original Data

10011001	11100010	00100100	10000100
----------	----------	----------	----------

Row parities

10011001	0
11100010	0
00100100	0
10000100	0
11011011	0

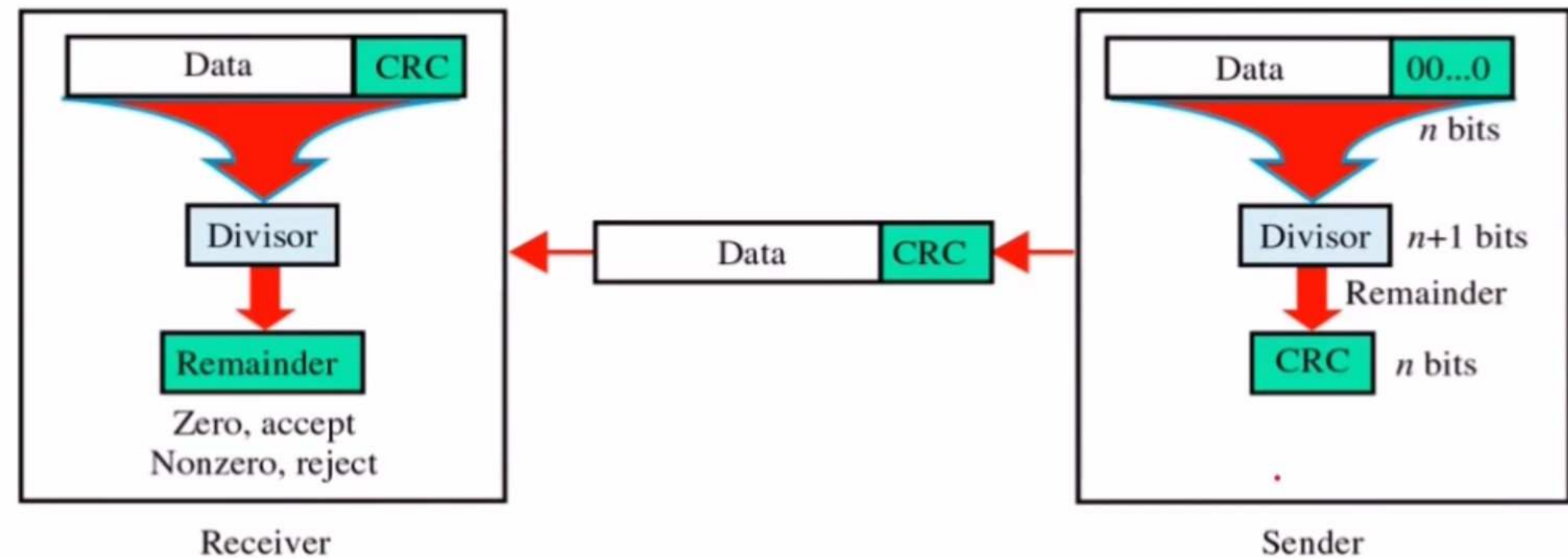
Column
parities →

100110010	111000100	001001000	100001000	110110110
-----------	-----------	-----------	-----------	-----------

Data to be sent

❑ CRC(Cyclic Redundancy Check)

~ is based on binary division.



One's Complement

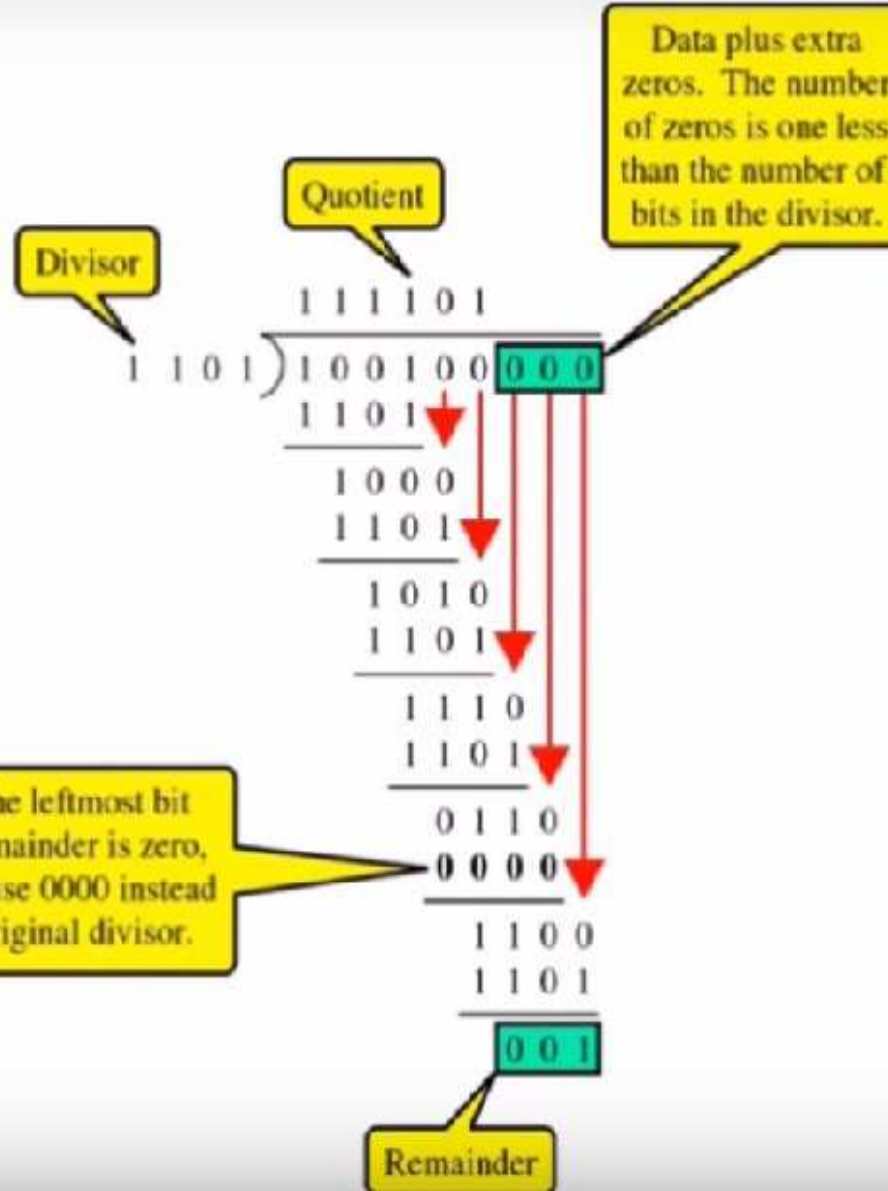
Invert all bits. Each 1 becomes a 0, and each 0 becomes a 1.

Original Value		One's Complement
0	→	1
1		0
1010	→	0101
1111		0000
11110000	→	00001111
10100011		01011100
11110000 10100101	→	00001111 01011010

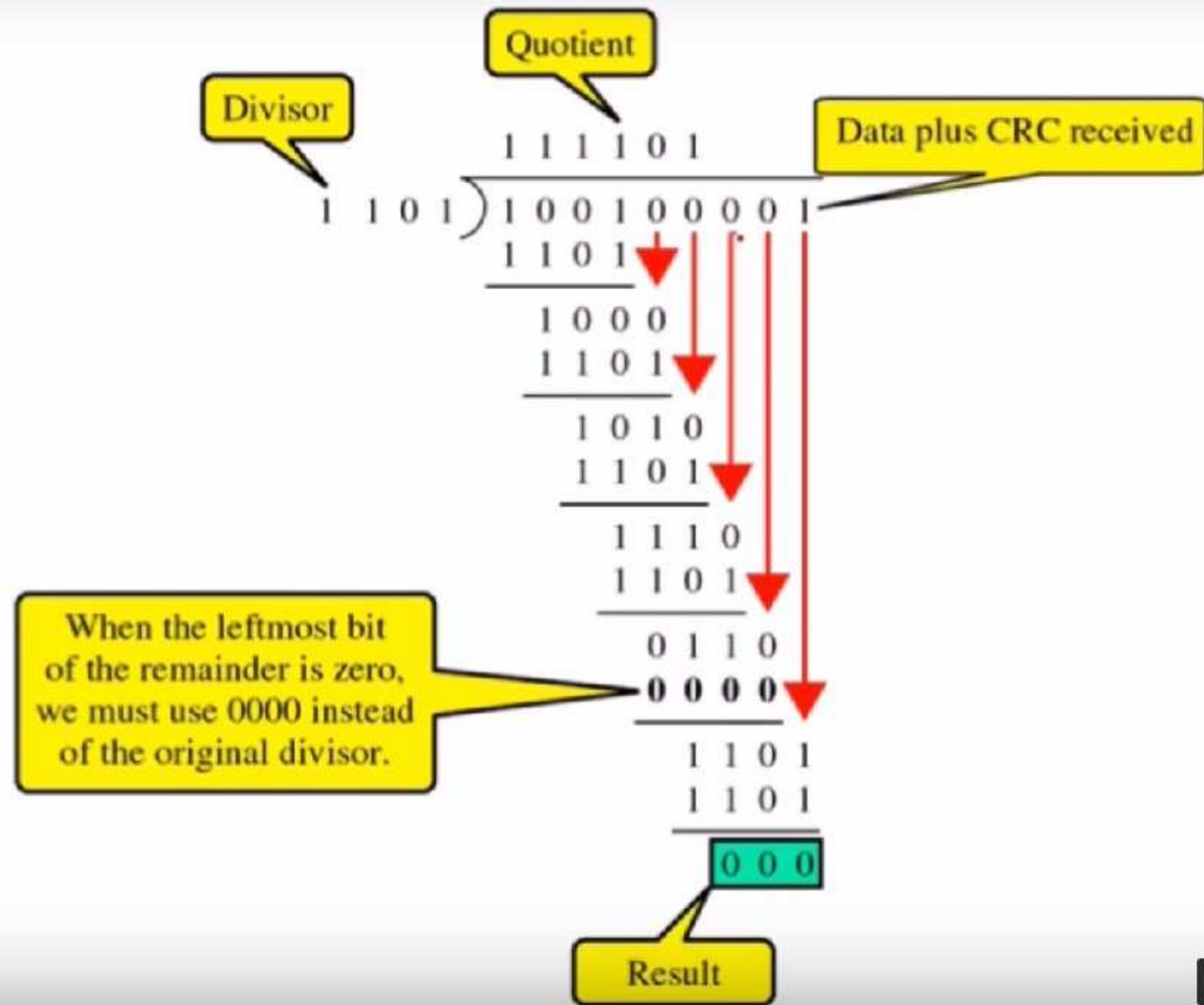
☐ CRC generator

~ uses modular-2 division.

Binary Division in a CRC Generator



Binary Division in a CRC Checker

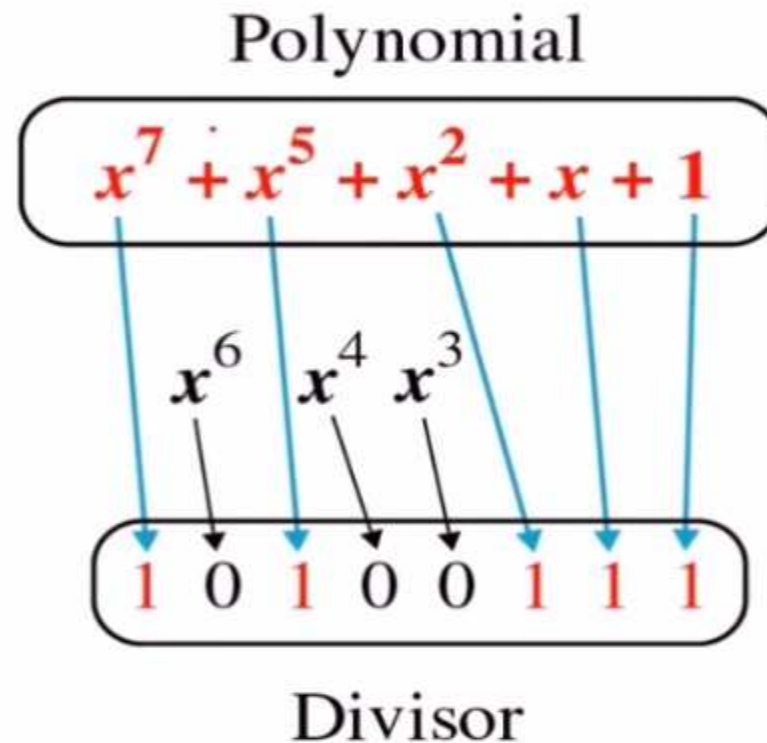


Polynomials

- ❖ CRC generator(divisor) is most often represented not as a string of 1s and 0s, but as an algebraic polynomial.

$$x^7 + x^5 + x^2 + x + 1$$

- ❑ A polynomial representing a divisor



Example-2

original message
1 0 1 0 0 0 0

@ means X-OR

Generator polynomial
 x^3+1
 $1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$
CRC generator
1 0 0 1 4-bit

If CRC generator is of n bit then append $(n-1)$ zeros in the end of original message

Sender

```

1001 | 1010000000
  @1001
  -----
0011000000
  @1001
  -----
01010000
  @1001
  -----
0011000
  @1001
  -----
01010
  @1001
  -----
0011
  
```

Message to be transmitted

```

1010000000
  + 011
  -----
1010000011
  
```

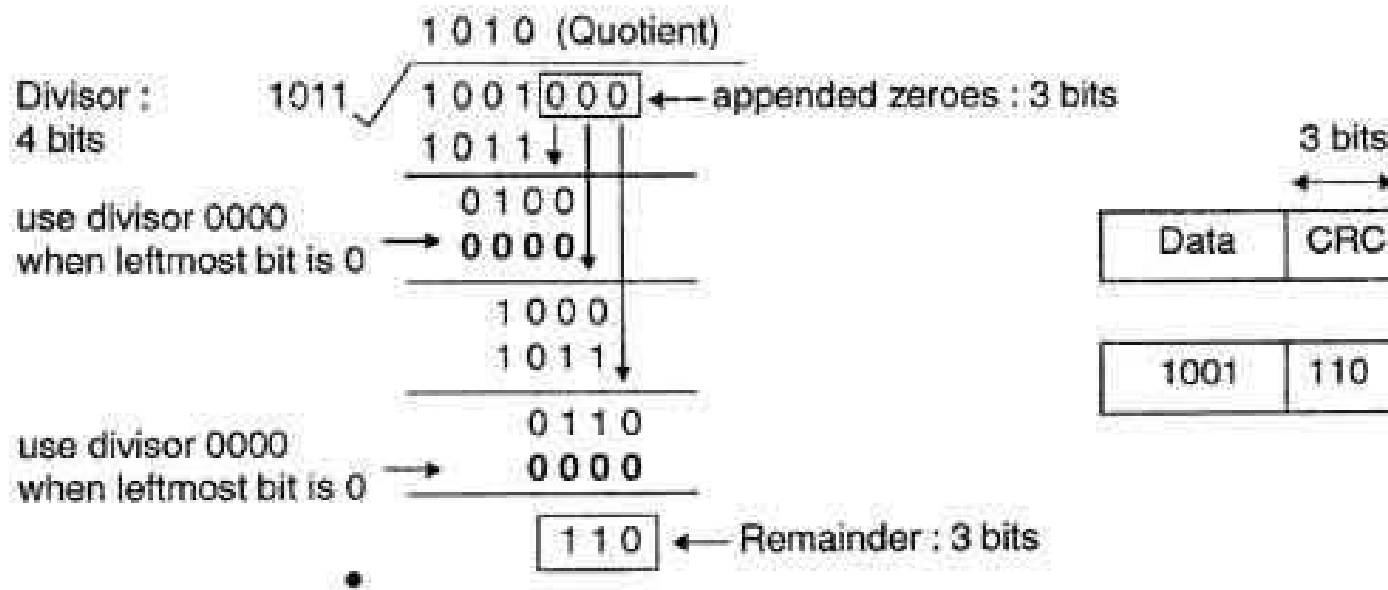
```

1001 | 1010000011
  @1001
  -----
0011000011
  @1001
  -----
01010011
  @1001
  -----
0011011
  @1001
  -----
01001
  @1001
  -----
0000
  
```

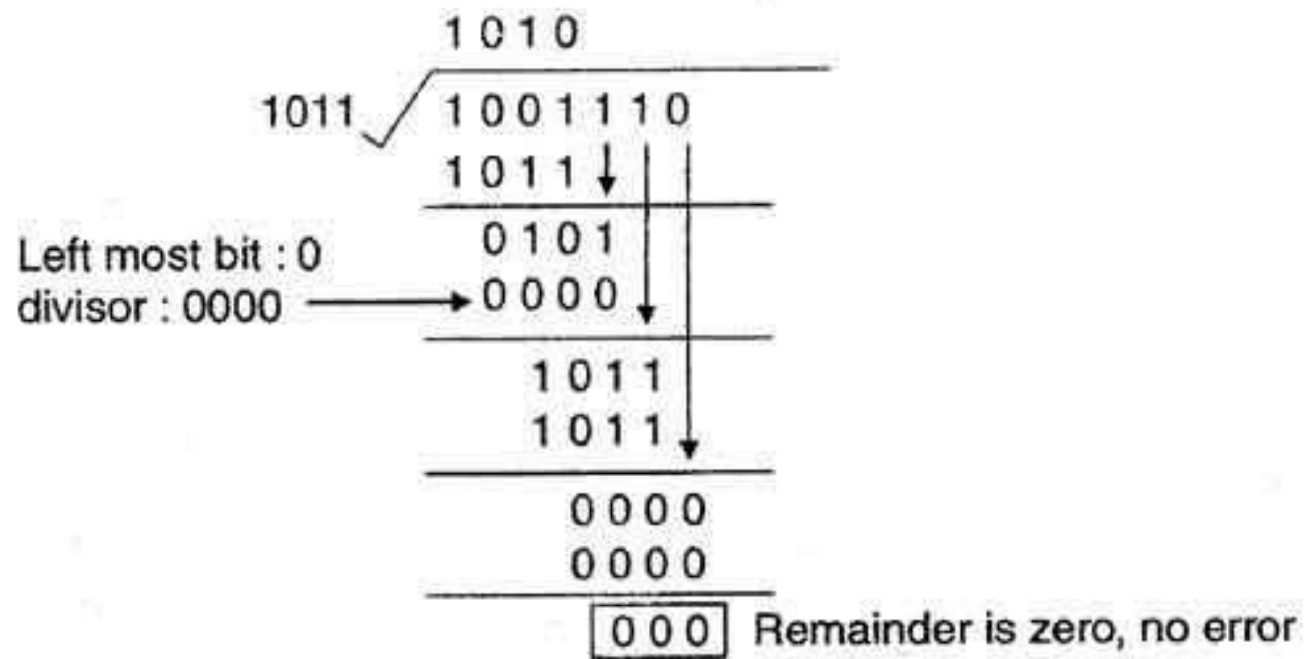
Receiver

Zero means data is accepted

Example-3 Data unit 1001000 is divided by 1011.



CRC generated (Binary division)



CRC decoded (binary division)

Consider the following message $M = 1010001101$. The cyclic redundancy check (CRC) for this message using the divisor polynomial $x^5 + x^4 + x^2 + 1$ is :

- (A) 01110
- (B) 01011
- (C) 10101
- (D) 10110

$$M = 1010001101$$

$$x^5 + x^4 + 0 \cdot x^3 + x^2 + 0 \cdot x + x^0$$

$$1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$$

appended $M = 101000110100000$

$$110101 \overline{) 101000110100000}$$

XOR \leftarrow \oplus

$$\begin{array}{r} 110101 \\ \oplus 011101110100000 \\ \hline 00111010100000 \\ \oplus 110101 \\ \hline 001111100000 \\ \oplus 110101 \\ \hline 0010110000 \\ \oplus 110101 \\ \hline 01100100 \\ \oplus 110101 \\ \hline 1110 \end{array}$$

$\boxed{1110}$
 \swarrow make it 5 digit

$\boxed{01110}$ Ans

Cyclic Redundancy Code (CRC)

A form of channel coding known as a *Cyclic Redundancy Code (CRC)* is used in high-speed data networks

Key properties of CRC are summarized below:

Arbitrary Length Message	As with a checksum, the size of a dataword is not fixed, which means a CRC can be applied to an arbitrary length message
Excellent Error Detection	Because the value computed depends on the sequence of bits in a message, a CRC provides excellent error detection capability
Fast Hardware Implementation	Despite its sophisticated mathematical basis, a CRC computation can be carried out extremely fast by hardware

CRC (cont'd)

Cyclic redundancy codes:

- Uses a mathematical function
- More complex to compute than checksums
- Handles more errors
- Used with frame transmission schemes
- A single set of check digits is generated and appended at the end of each frame transmitted
- In this method the bits of data are treated as coefficients of a polynomial

CRC (cont'd)

CRC Coding:

A k -bit message is regarded as the coefficient list for a polynomial with k terms, ranging from $x^{(k-1)}$ to x^0 . The high-order (leftmost) bit is the coefficient of $x^{(k-1)}$; the next bit is the coefficient of $x^{(k-2)}$, and so on.

The check digits are generated by multiplying the k -bit message by x^n and dividing the product by an $(n+1)$ -bit code polynomial (modulo 2). The n -bit remainder is used as the check digits.

Decoding:

The complete received bit sequence is divided by the same generator polynomial (modulo 2).

If the remainder is zero, no errors occurred.

If the remainder is nonzero, a transmission error has occurred.

CRC Calculation

This figure shows the division of 1010 ($k=4$) which represents the data by a constant generator function: 1011 ($n+1=4$).

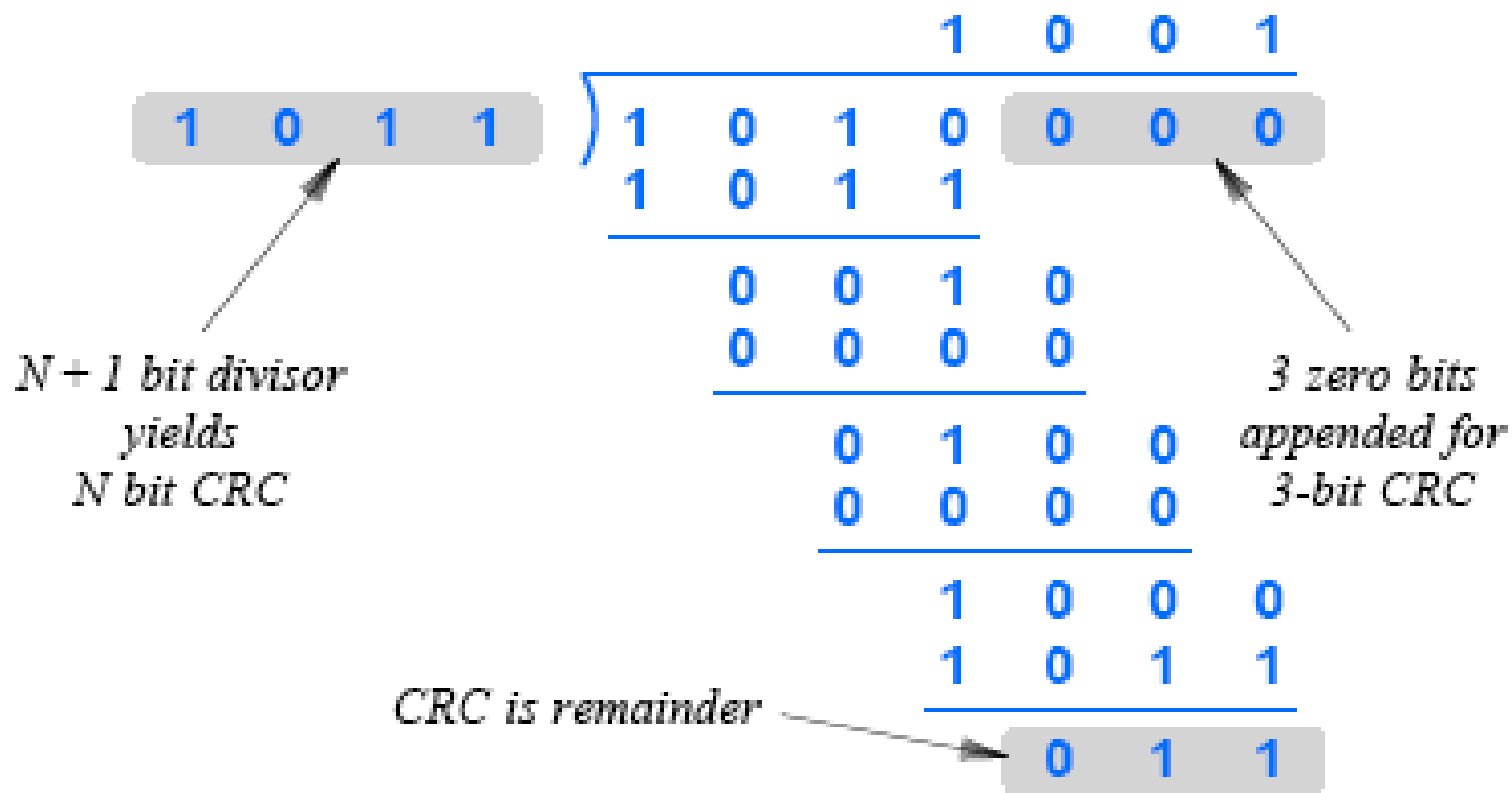


Figure 8.12

CRC (cont'd)

A generator polynomial of $N+1$ bits will detect:

- all single-bit errors
- all double-bit errors
- all odd-number of bit errors
- all error bursts $< N+1$ bits
- most error bursts $\geq N+1$ bits

CRCs and Polynomial Representation

We can view the above process as a **polynomial division**:

Think of each bit in a binary number as the coefficient of a term in a polynomial

For example, we can think of the divisor in Figure 8.12, 1011, as coefficients in the following polynomial:

$$1 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0 = x^3 + x + 1$$

Similarly, the dividend in Figure 8.12, 1010000, represents the polynomial:

$$x^6 + x^4$$

Code Polynomials (or generator polynomials):

Code polynomials are usually of degree 12 or 16 or 32

Six polynomials are in widespread use:

- Ethernet and FDDI use CRC-32
- HDLC uses CRC-CCITT
- ATM uses CRC-8 and CRC-10
- Three polynomials are international standards: CRC-12, CRC-16, and CRC-CCITT

CRC	$C(x)$
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} +$ $x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Test yourself

Hamming Code Numerical Questions as available in End term Question Paper

Error Correction Techniques

Error correction techniques find out the exact number of bits that have been corrupted and as well as their locations. There are two principle ways

- **Backward Error Correction (Retransmission)** – If the receiver detects an error in the incoming frame, it requests the sender to retransmit the frame. It is a relatively simple technique. But it can be efficiently used only where retransmitting is not expensive as in fiber optics and the time for retransmission is low relative to the requirements of the application.
- **Forward Error Correction** – If the receiver detects some error in the incoming frame, it executes error-correcting code that generates the actual frame. This saves bandwidth required for retransmission. It is inevitable in real-time systems. However, if there are too many errors, the frames need to be retransmitted.
- Hamming Codes

Checksum

In this error detection scheme, the following procedure is applied

- Data is divided into fixed sized frames or segments.
- The sender adds the segments using 1's complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
- The receiver adds the incoming segments along with the checksum using 1's complement arithmetic to get the sum and then complements it.
- If the result is zero, the received frames are accepted; otherwise, they are discarded.

Internet Checksum Algorithm

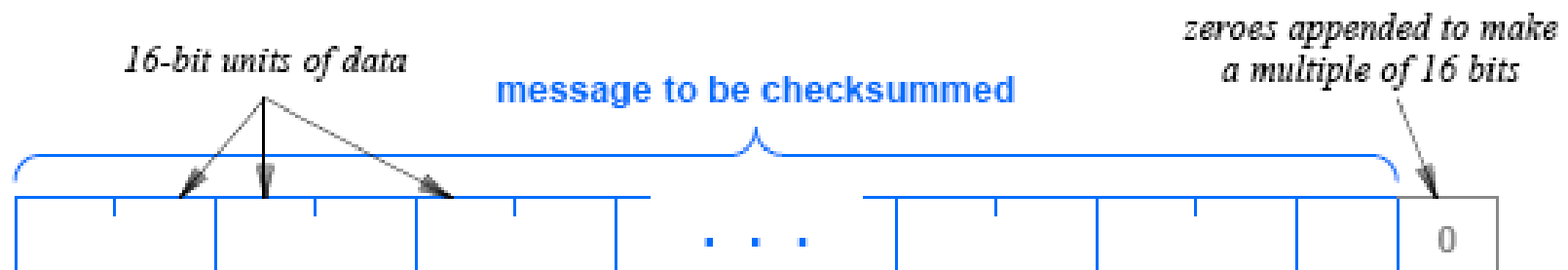
The Internet checksum is a simple error detection technique used by TCP/IP.

The Internet Checksum Algorithm is simple:

- treat the data being transmitted as 16-bit integers,
- add them together using 16-bit ones-complement arithmetic,
- take the complement of the sum as the checksum,
- send the checksum across the network with the original data.

The Internet checksum:

- Does not have strong error detection properties, but handles many multiple bit errors
- Cannot handle all errors
- It is easy to implement in software
- It is used in an end-to-end manner, so lower layer protocols catch most of the errors



Internet Checksum Algorithm (Cont'd)

The Internet checksum:

- does not have strong error detection properties, but handles many multiple bit errors
- Cannot handle all errors
- is easy to implement in software
- is used in a end-to-end manner, so lower layer protocols catch most of the errors

Hamming Code

Hamming code is a technique developed by R.W. Hamming for error correction. This method corrects the error by finding the state at which the error has occurred.

Determining the positions of redundancy bits

Till now, we know the exact number of redundancy bits required to be embedded with the particular data unit.

We know that to detect errors in a 7 bit code, 4 redundant bits are required.

Now, the next task is to determine the positions at which these redundancy bits will be placed within the data unit.

- These redundancy bits are placed at the positions which correspond to the power of 2.
- For example in case of 7 bit data, 4 redundancy bits are required, so making total number of bits as 11. The redundancy bits are placed in position 1, 2, 4 and 8 as shown in fig.



redundancy bits

Positions of redundancy bits in Hamming Code

Generating parity

- In Hamming code, each r bit is the VRC for one combination of data bits. r_1 is the VRC bit for one combination of data bits, r_2 is the VRC for another combination of data bits and so on.
- Each data bit may be included in more than one VRC calculation.
- r_1 bit is calculated using all bits positions whose binary representation includes a 1 in the rightmost position.
- r_2 bit calculated using all the bit positions with a 1 in the second position and so on.
- Therefore the various r bits are parity bits for different combination of bits.

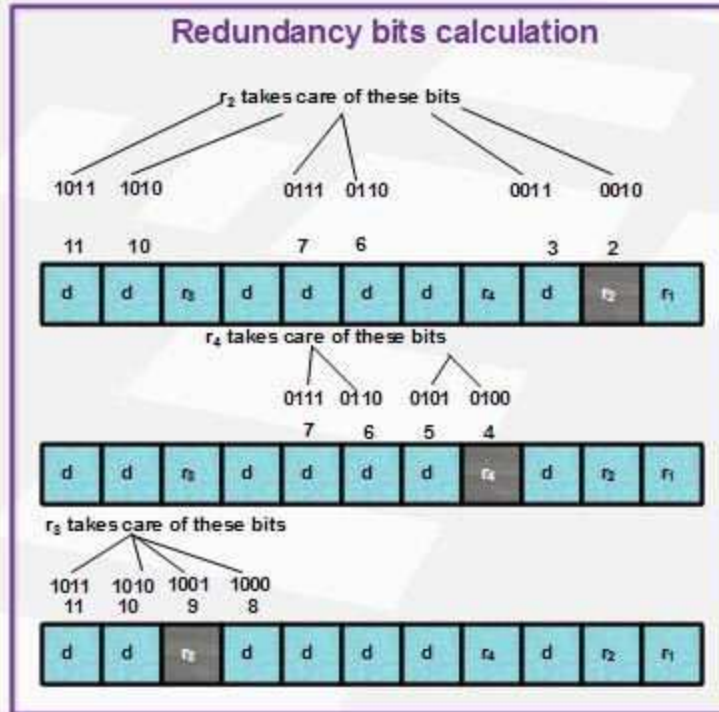
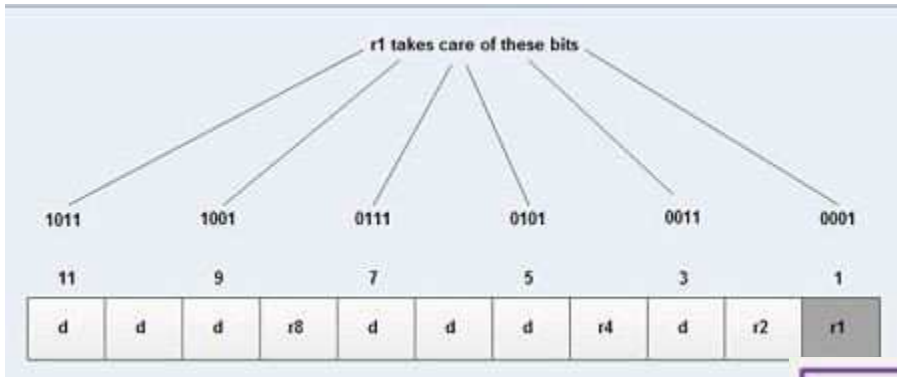
The various combinations are:

r_1 : bits 1,3,5, 7, 9, 11

r_2 : bits 2, 3, 6, 7, 10, 11

r_4 : bits 4, 5, 6, 7

r_8 : bits 8, 9, 10, 11



ALOHA

- ALOHA is a system for coordinating and arbitrating access to a shared communication Networks channel.
- It was developed in the 1970s by Norman Abramson and his colleagues at the University of Hawaii.
- The original system used for ground based radio broadcasting, but the system has been implemented in satellite communication systems.

ALOHA

- A shared communication system like ALOHA requires a method of handling collisions that occur when two or more systems attempt to transmit on the channel at the same time.
- In the ALOHA system, a node transmits whenever data is available to send. If another node transmits at the same time, a collision occurs, and the frames that were transmitted are lost.
- However, a node can listen to broadcasts on the medium, even its own, and determine whether the frames were transmitted.

- **Aloha means "Hello"**. Aloha is a multiple access protocol at the datalink layer and proposes how multiple terminals access the medium without interference or collision.
- In 1972 Roberts developed a protocol that would increase the capacity of aloha two fold.
- The Slotted Aloha protocol involves dividing the time interval into discrete slots and each slot interval corresponds to the time period of one frame.
- This method requires synchronization between the sending nodes to prevent collisions.

PURE ALOHA

- In pure ALOHA, the stations transmit frames whenever they have data to send.
- When two or more stations transmit simultaneously, there is collision and the frames are destroyed.
- In pure ALOHA, whenever any station transmits a frame, it expects the acknowledgement from the receiver.
- If acknowledgement is not received within specified time, the station assumes that the frame (or acknowledgement) has been destroyed.
- If the frame is destroyed because of collision the station waits for a random amount of time and sends it again. This waiting time must be random otherwise same frames will collide again and again.
- Therefore pure ALOHA dictates that when time-out period passes, each station must wait for a random amount of time before resending its frame. This randomness will help avoid more collisions.

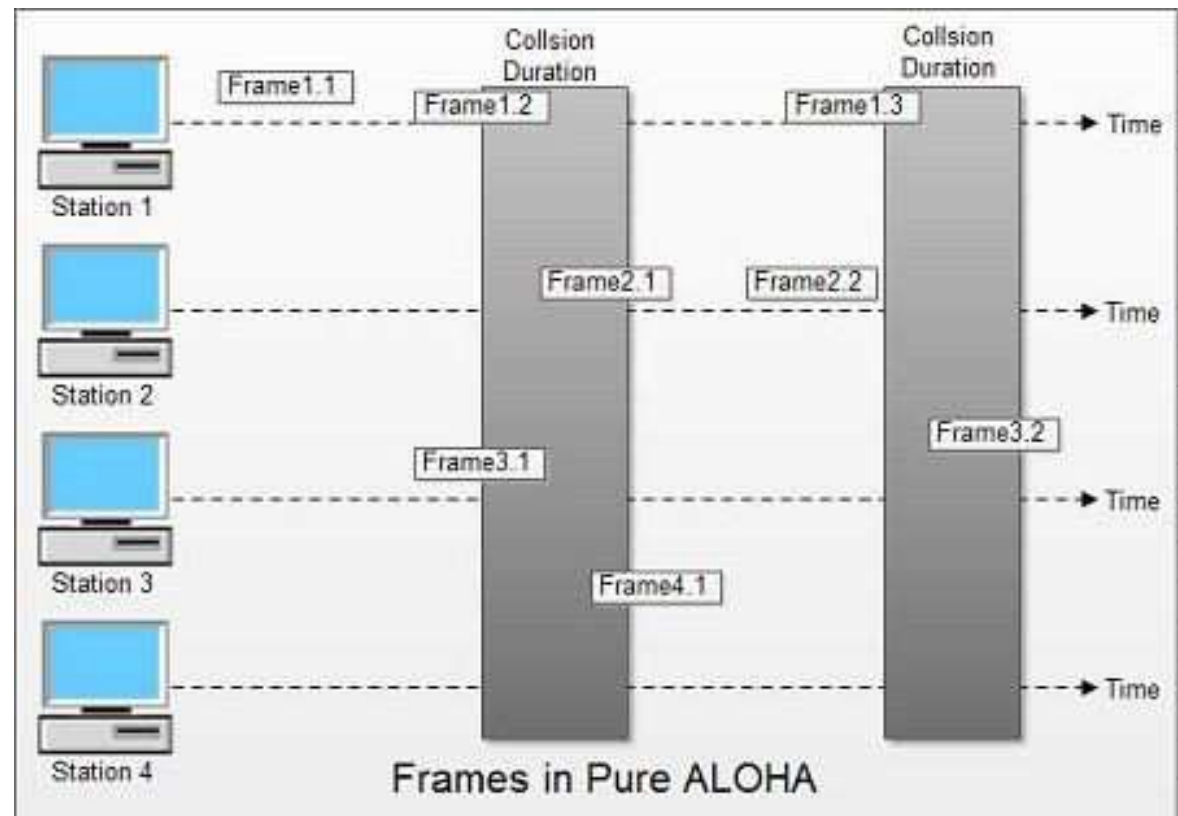
PURE ALOHA

- Figure shows an example of frame collisions in pure ALOHA.
- In fig there are four stations that contend with one another for access to shared channel.

All these stations are transmitting frames. Some of these frames collide because multiple frames are in contention for the shared channel.

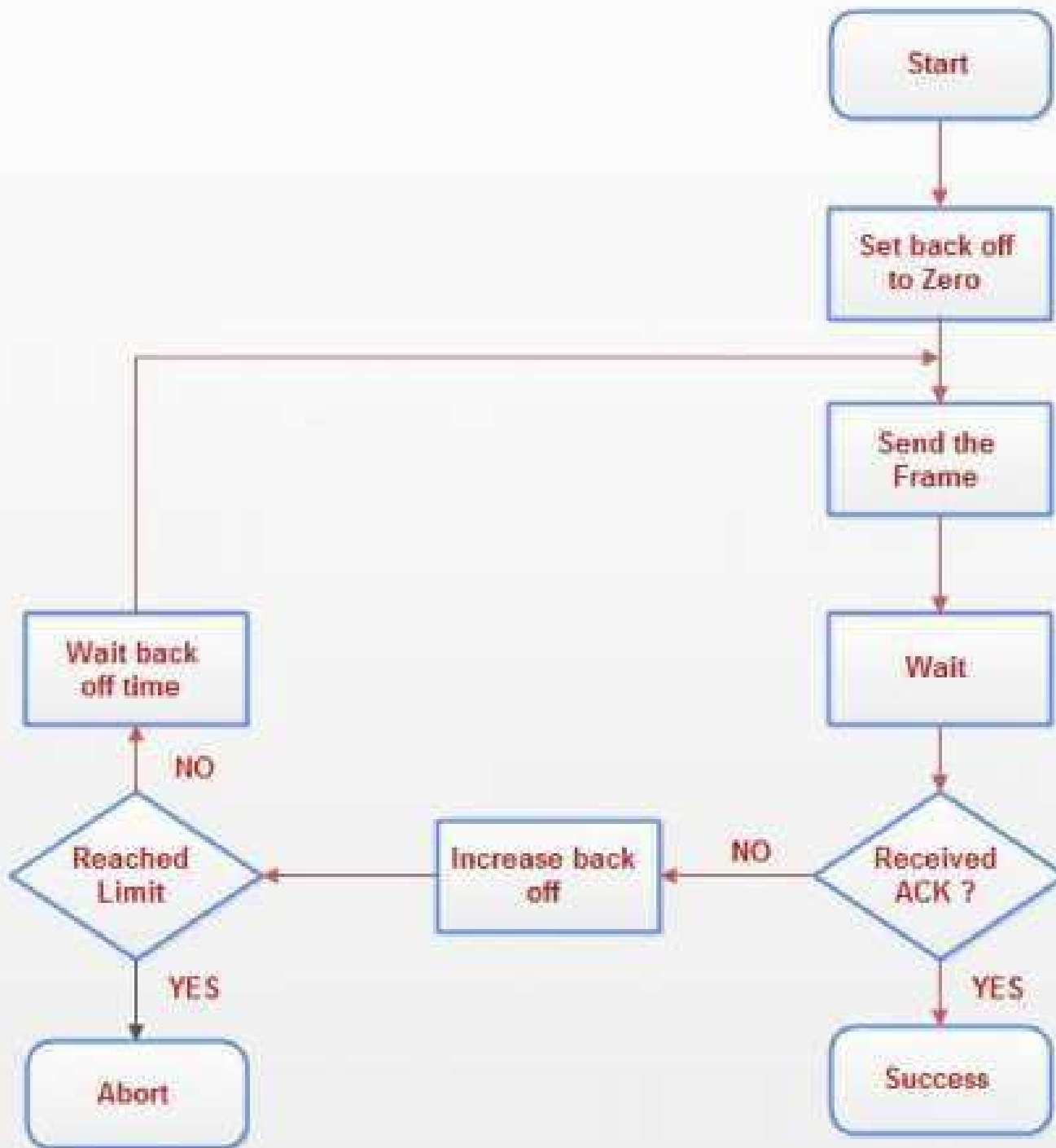
Only two frames, frame 1.1 and frame 2.2 survive. All other frames are destroyed.

- Whenever two frames try to occupy the channel at the same time, there will be a collision and both will be damaged. **If first bit of a new frame overlaps with just the last bit of a frame almost finished, both frames will be totally destroyed and both will have to be retransmitted**



Slotted ALOHA

- Slotted ALOHA was invented to improve the efficiency of pure ALOHA as chances of collision in pure ALOHA are very high.
- In slotted ALOHA, the time of the shared channel is divided into discrete intervals called
slots.
- The stations can send a frame only at the beginning of the slot and only one frame is sent in each slot.
- In slotted ALOHA, if any station is not able to place the frame onto the channel at the beginning of the slot *i.e.* it misses the time slot then the station has to wait until the beginning of the next time slot.
- Slotted ALOHA still has an edge over pure ALOHA as chances of collision are reduced to one-half.



Protocol flow chart for ALOHA

Explanation:

- A station which has a frame ready will send it.
- Then it waits for some time.
- If it receives the acknowledgement then the transmission is successful.
- Otherwise the station uses a backoff strategy, and sends the packet again.
- After many times if there is no acknowledgement then the station aborts the idea of transmission.

Flow and Error Control

Flow Control

- Flow control coordinates the amount of data that can be sent before receiving acknowledgement
- It is one of the most important functions of data link layer.
- Flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgement from the receiver.
- Receiver has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data.
- Receiver must inform the sender before the limits are reached and request that the transmitter to send fewer frames or stop temporarily.
- Since the rate of processing is often slower than the rate of transmission, receiver has a block of memory (buffer) for storing incoming data until they are processed.

Error Control

- Error control includes both error detection and error correction.
- It allows the receiver to inform the sender if a frame is lost or damaged during transmission and coordinates the retransmission of those frames by the sender.
- Error control in the data link layer is based on automatic repeat request (ARQ).
Whenever an error is detected, specified frames are retransmitted.

Error and Flow Control Mechanisms

- Stop-and-Wait
- Go-Back-N ARQ
- Selective-Repeat ARQ

Stop-and-wait ARQ.

- **Stop-and-wait ARQ**, also referred to as alternating bit **protocol**, is a method in telecommunications to send information between two connected devices.
- It ensures that information is not lost due to dropped packets and that packets are received in the correct order

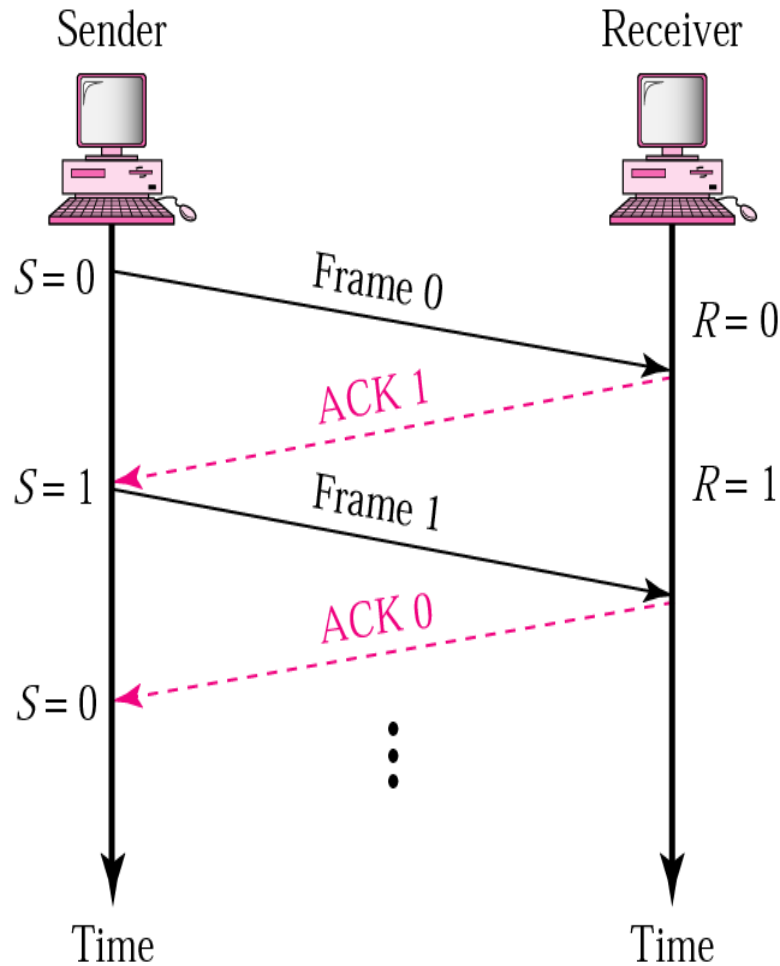
ARQ

- Automatic repeat request (**ARQ**) is a **protocol** for error control in data transmission.
- When the receiver detects an error in a packet, it automatically requests the transmitter to resend the packet.
- **ARQ** is sometimes used with Global System for Mobile (GSM) communication to guarantee data integrity.

Concept behind Stop and Wait ARQ

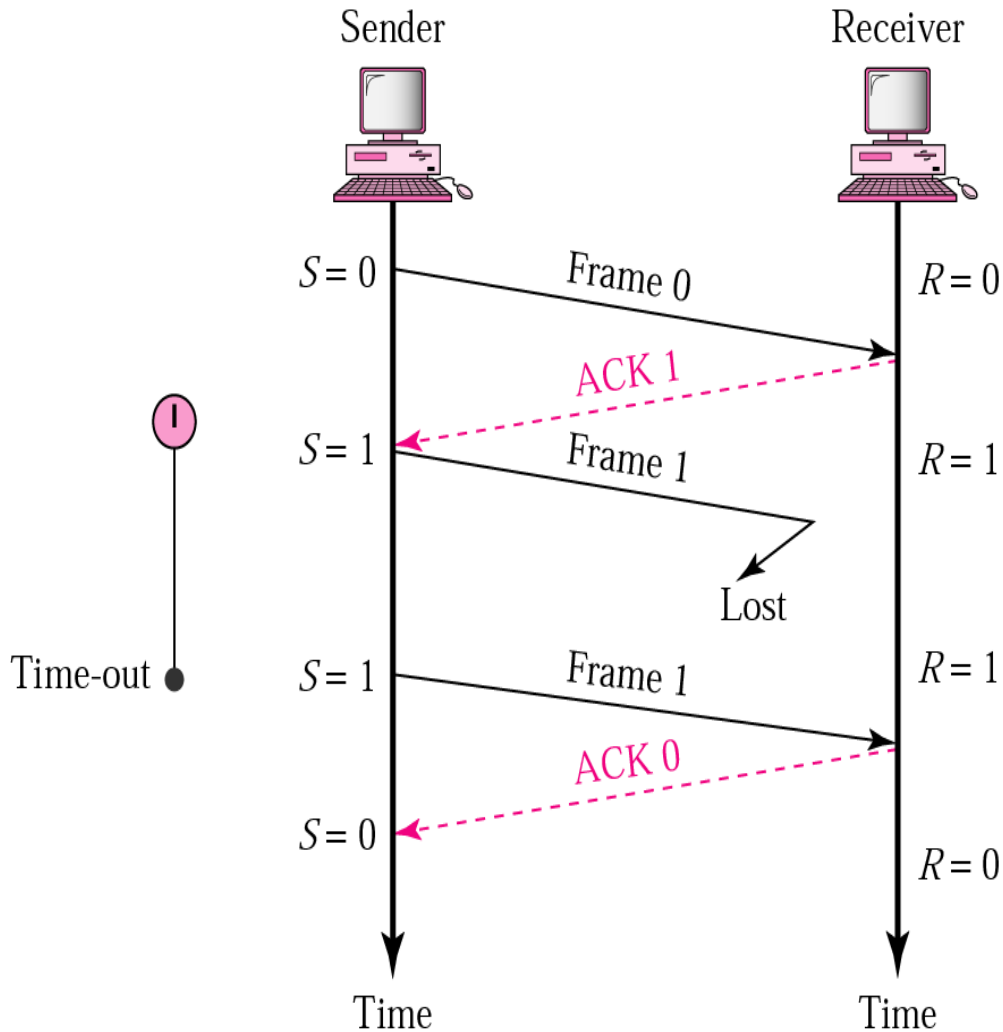
- **Stop-and-wait flow control** is the simplest form of **flow control**.
- In this method, the receiver indicates its readiness to receive data for each frame, the message is broken into multiple frames.
- The sender waits for an **ACK** (acknowledgement) after every frame for specified time (called time out).

Stop-and-Wait



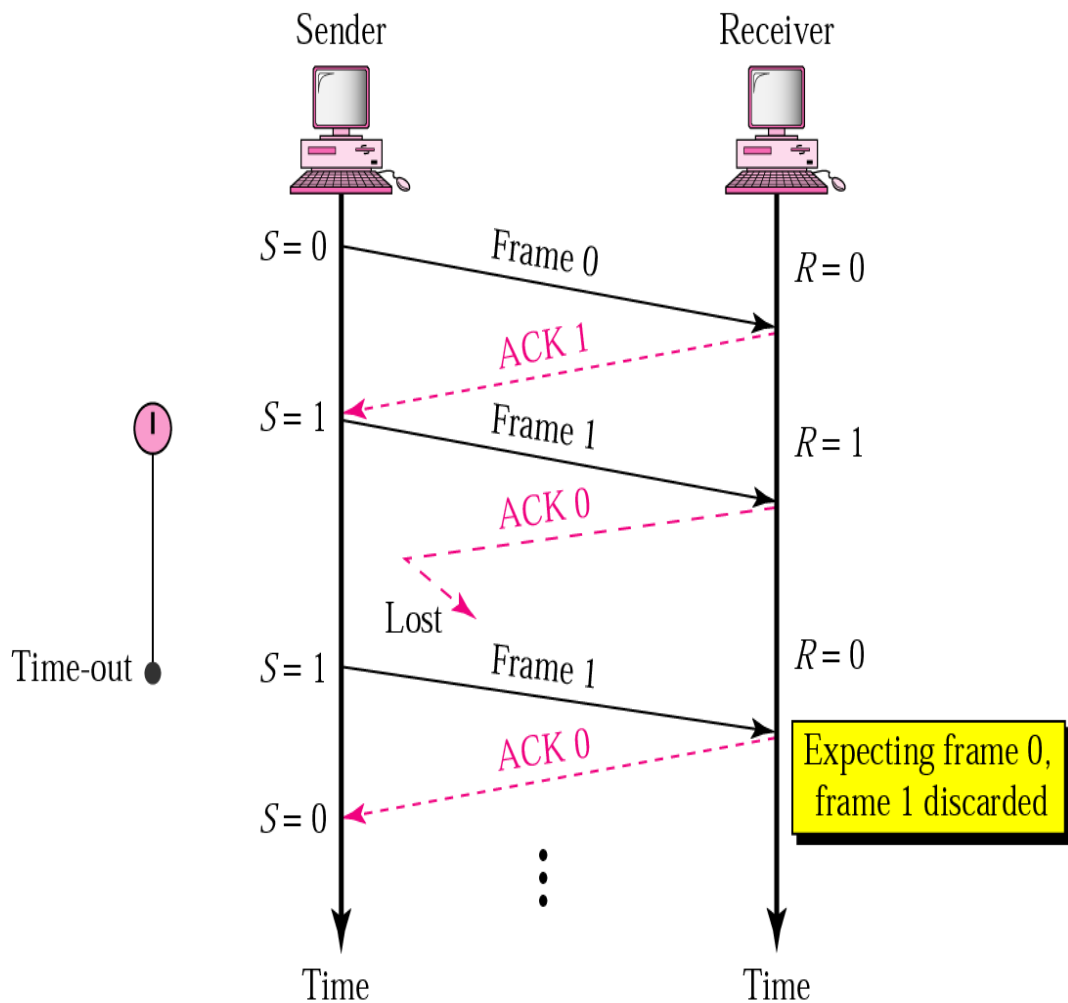
- Sender keeps a copy of the last frame until it receives an acknowledgement.
- For identification, both data frames and acknowledgements (ACK) frames are numbered alternatively 0 and 1.
- Sender has a control variable (S) that holds the number of the recently sent frame. (0 or 1)
- Receiver has a control variable R that holds the number of the next frame expected (0 or 1).
- Sender starts a timer when it sends a frame. If an ACK is not received within a allocated time period, the sender assumes that the frame was lost or damaged and resends it
- Receiver send only positive ACK if the frame is intact.
- ACK number always defines the number of the next expected frame

Stop-and-Wait ARQ, lost ACK frame



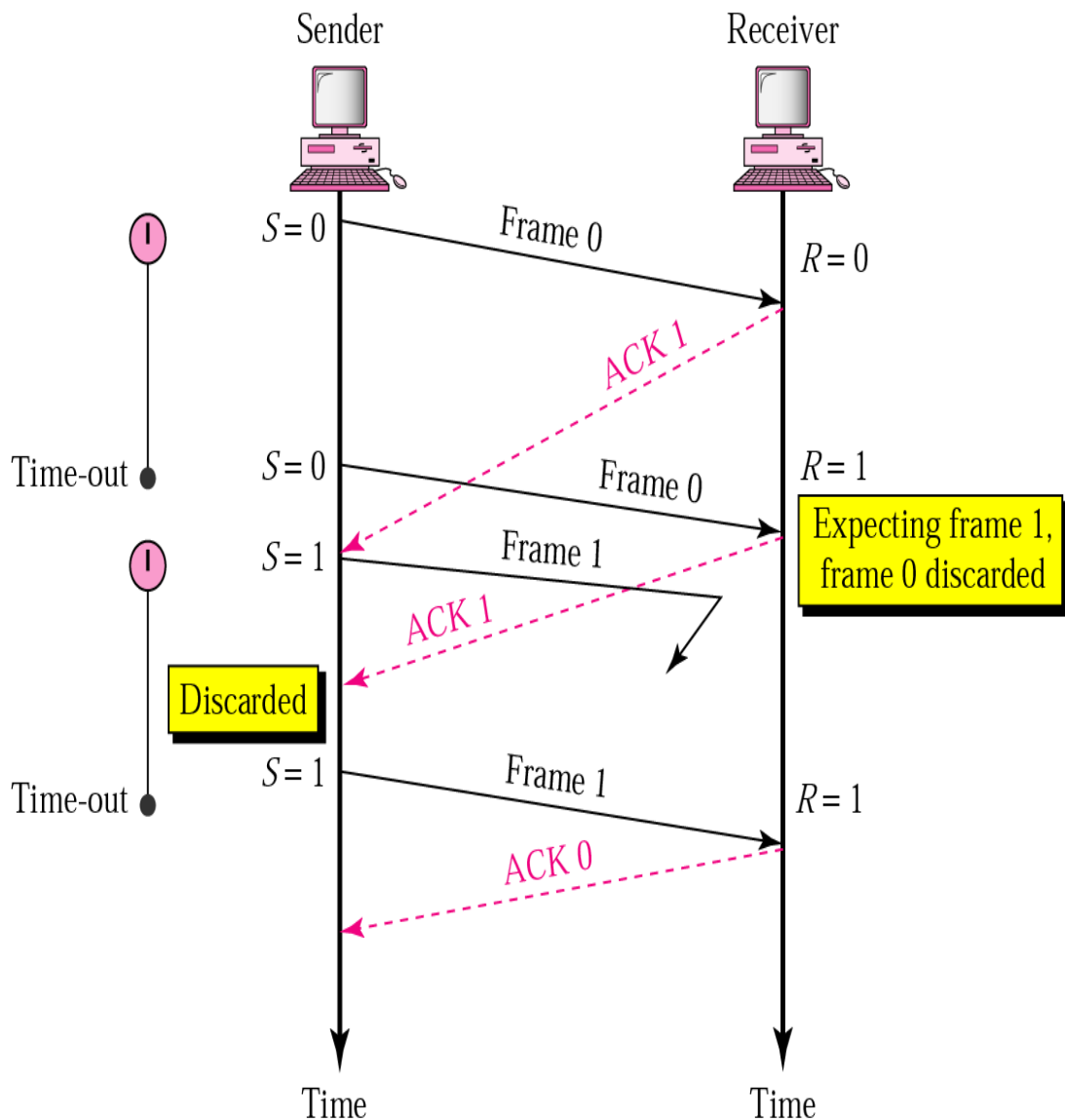
- When a receiver receives a damaged frame, it discards it and keeps its value of R .
- After the timer at the sender expires, another copy of frame 1 is sent.

Stop-and-wait, lost ACK frame



- If the sender receives a damaged ACK, it discards it.
- When the timer of the sender expires, the sender retransmits frame 1.
- Receiver has already received frame 1 and expecting to receive frame 0 ($R=0$). Therefore it discards the second copy of frame 1.

Stop-and-wait, delayed ACK frame



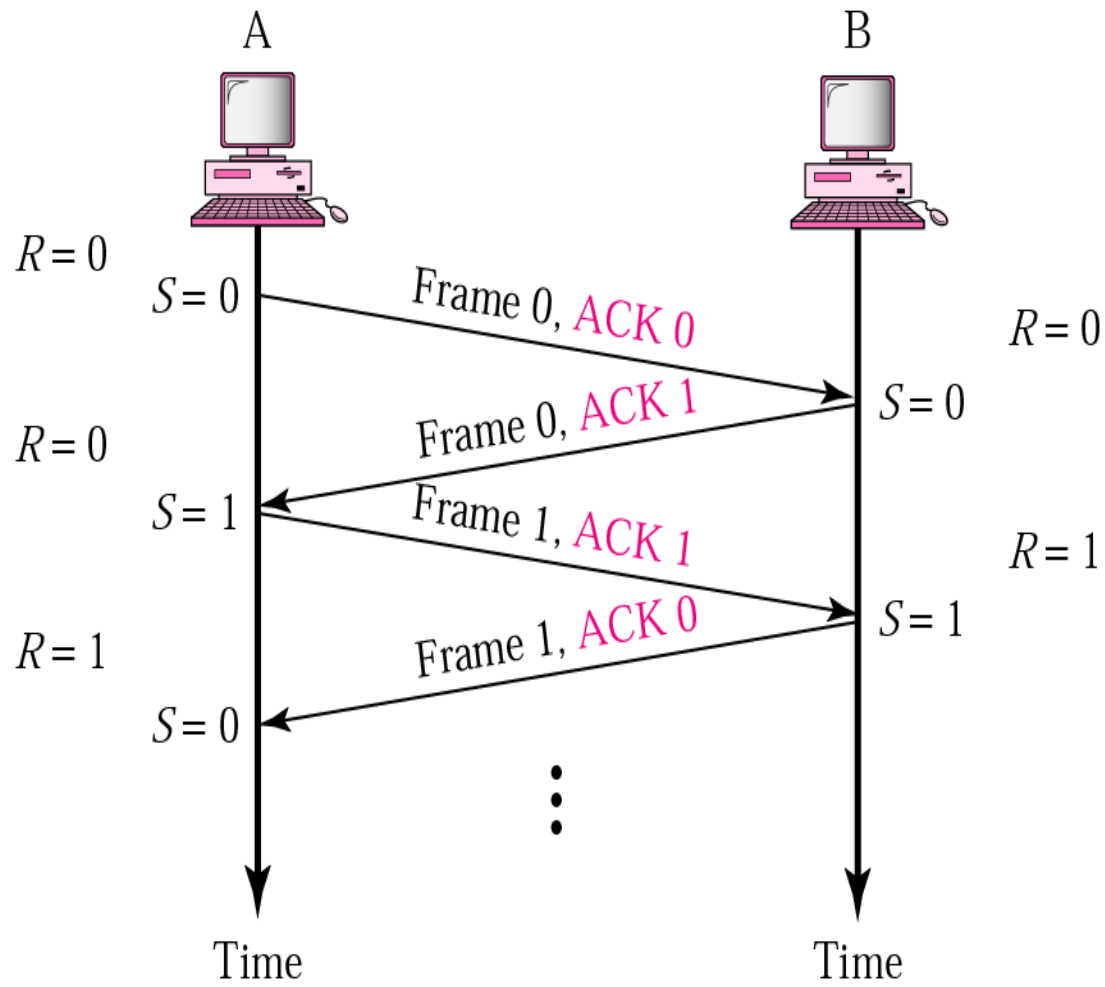
- The ACK can be delayed at the receiver or due to some problem
- It is received after the timer for frame 0 has expired.
- Sender retransmitted a copy of frame 0. However, $R = 1$ means receiver expects to see frame 1. Receiver discards the duplicate frame 0.
- Sender receives 2 ACKs, it discards the second ACK.

• Piggybacking

(1) Gaining access to a restricted communications channel by using the session another user already established.

(2) Piggybacking can be defeated by logging out before walking away from a workstation or terminal or by initiating a screensaver that requires re-authentication when resuming.

Piggybacking



- A method to combine a data frame with ACK.
- Station A and B both have data to send.
- Instead of sending separately, station A sends a data frame that includes an ACK.
- Station B does the same thing.
- Piggybacking saves bandwidth.

Disadvantage of Stop-and-Wait

- In stop-and-wait, at any point in time, there is only one frame that is sent and waiting to be acknowledged.
- This is not a good use of transmission medium.
- To improve efficiency, multiple frames should be in transition while waiting for ACK.
- Two protocols use the above concept,
 - **Go-Back-N ARQ**
 - **Selective Repeat ARQ**

Go-Back-N ARQ

- **Go-Back-N ARQ** is a specific instance of the automatic repeat request (ARQ)**protocol**, in which the sending process continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver.
- It can transmit **N** frames to the peer before requiring an ACK.

Go-Back-N ARQ

- We can send up to W frames before worrying about ACKs.
- We keep a copy of these frames until the ACKs arrive.
- This procedure requires additional features to be added to Stop-and-Wait ARQ.

Go-Back-N ARQ (Sliding Window Protocol)

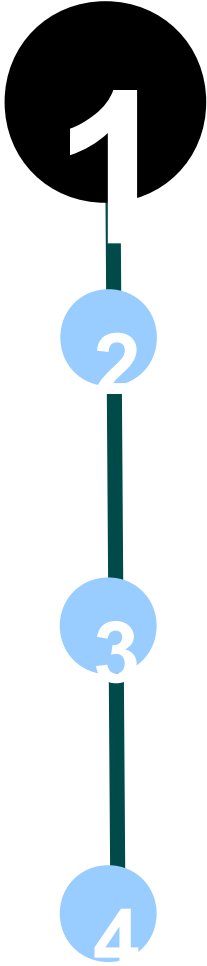
Go-Back-N is an instance of the Automatic Repeat-request (ARQ) Protocol, in which the sending process continues to send a number of frames specified by a window size even without receiving an ACK packet from the receiver.

The receiver process keeps track of the sequence number of the next frame it expects, and sends it with every ACK it sends.

If a frame from the sender does not reach the receiver, the receiver will stop acknowledging received frames.

The sender after sending all of the frames will detect that all of the frames since the first lost frame are outstanding, and will go back to sequence number of the last ACK it received and fill its window starting with that frame and continue the process over again.

Understanding Go-Back-N ARQ:



Go-Back-N ARQ is a specific instance of automatic repeat request (ARQ) protocol which is an error recovery mechanism to provide reliability. objects with a span smaller than the structuring element.

It is a special type of sliding window protocol.

In this the sending process continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver. removes perimeter pixels from larger image objects.

In Go-Back-N ARQ, the size of the sender window must be less than N and the size of the receiver window is always 1.

When the frame is damaged the sender goes back and sends a set of frames starting from the last one ACKn'd.

The number of retransmitted frames is N .

Constraints taken while using Go-Back-N ARQ:

1. Sequence Number
2. Sender and Receiver Sliding Window
3. Acknowledgment
4. Resending Frames

Sender's Action:

Once the sender has sent all of the frames in its window, it will detect that all of the frames since the first lost frame are outstanding, and will go back to sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

Receiver's Action:

1. The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends.
2. The receiver will ignore any frame that does not have the exact sequence number it expects – whether that frame is a 'past' duplicate of a frame it has already ACK'ed or whether that frame is a 'future' frame past the last packet it is waiting for.

Choosing a Window size (N):

There are a few things to keep in mind when choosing a value for N.

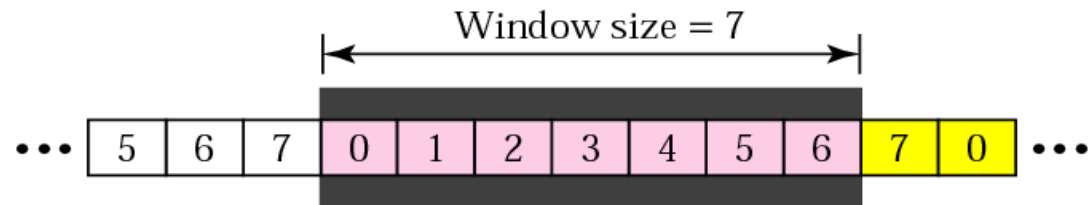
1. The sender must not transmit too fast. N should be bounded by the receiver's ability to process packets.
2. N must be smaller than the number of sequence numbers (if they are numbered from zero to N) to verify transmission in cases of any packet (any data or ACK packet) being dropped.
3. Given the bounds presented in (1) and (2), choose N to be the largest number possible.

Sequence Numbers

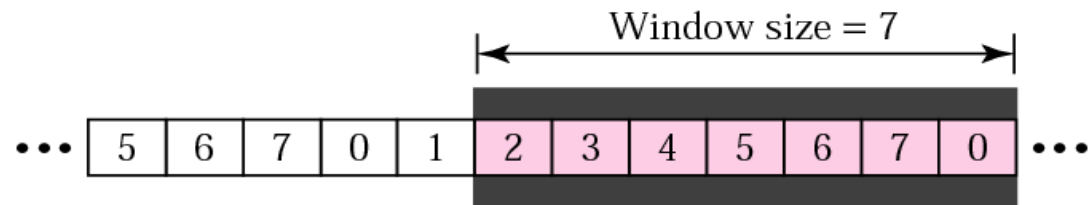
- Frames from a sender are numbered sequentially.
- We need to set a limit since we need to include the sequence number of each frame in the header.
- If the header of the frame allows m bits for sequence number, the sequence numbers range from 0 to $2^m - 1$. for $m = 3$, sequence numbers are: 1, 2, 3, 4, 5, 6, 7.
- We can repeat the sequence number.
- Sequence numbers are:
0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, ...

Sender Sliding Window

- At the sending site, to hold the outstanding frames until they are acknowledged, we use the concept of a window.
- The size of the window is at most $2^m - 1$ where m is the number of bits for the sequence number.
- Size of the window can be variable, e.g. TCP.
- The window slides to include new unsent frames when the



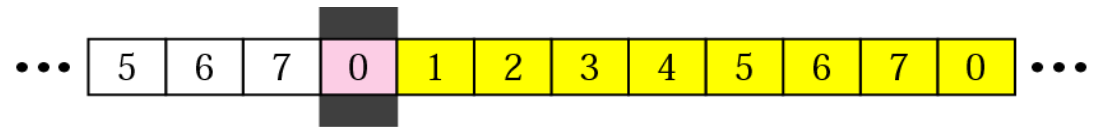
a. Before sliding



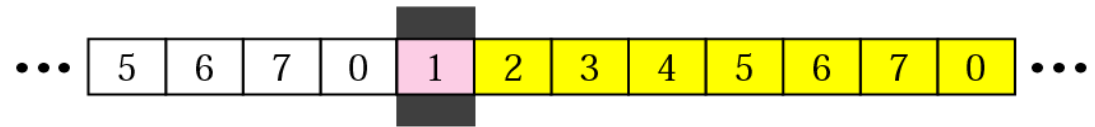
b. After sliding two frames

Receiver Sliding Window

- Size of the window at the receiving site is always 1 in this protocol.
- Receiver is always looking for a specific frame to arrive in a specific order.
- Any frame arriving out of order is discarded and needs to be resent.
- Receiver window slides as shown in fig. Receiver is waiting for frame 0 in part a.



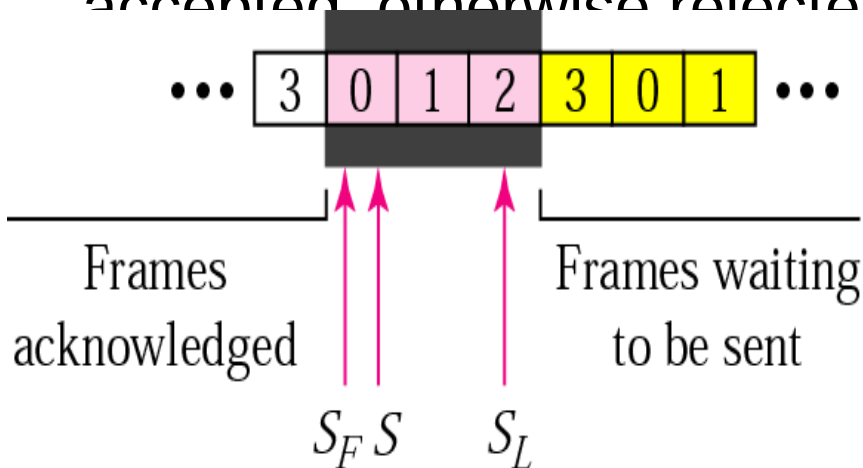
a. Before sliding



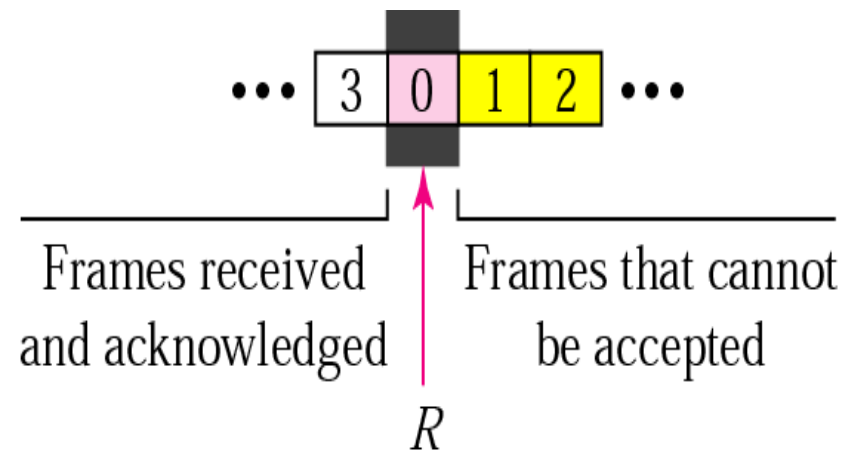
b. After sliding

Control Variables

- Sender has 3 variables: S , S_F , and S_L
- S holds the sequence number of recently sent frame
- S_F holds the sequence number of the first frame
- S_L holds the sequence number of the last frame
- Receiver only has the one variable, R , that holds the sequence number of the frame it expects to receive. If the seq. no. is the same as the value of R , the frame is accepted, otherwise rejected



a. Sender window



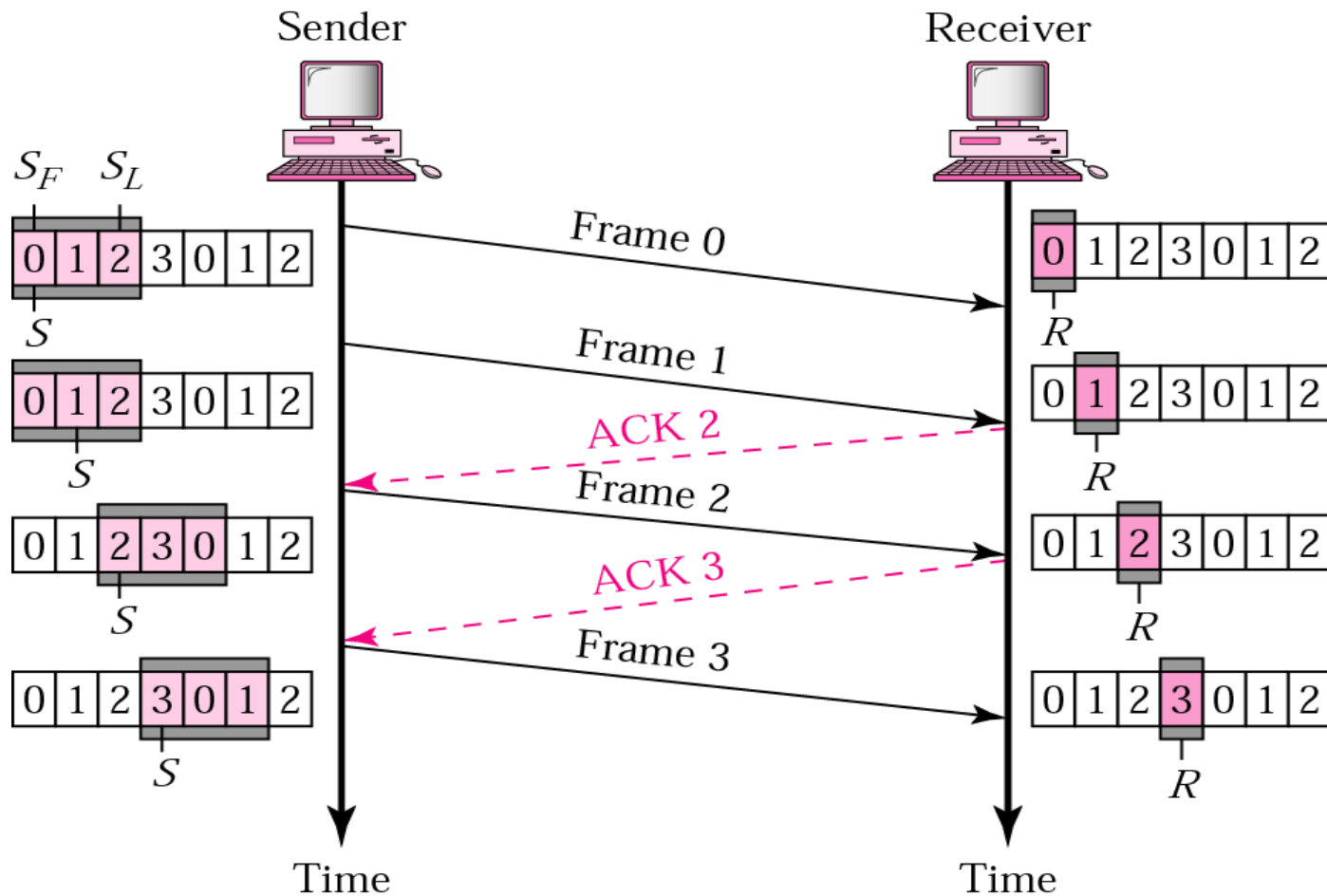
b. Receiver window

Acknowledgement

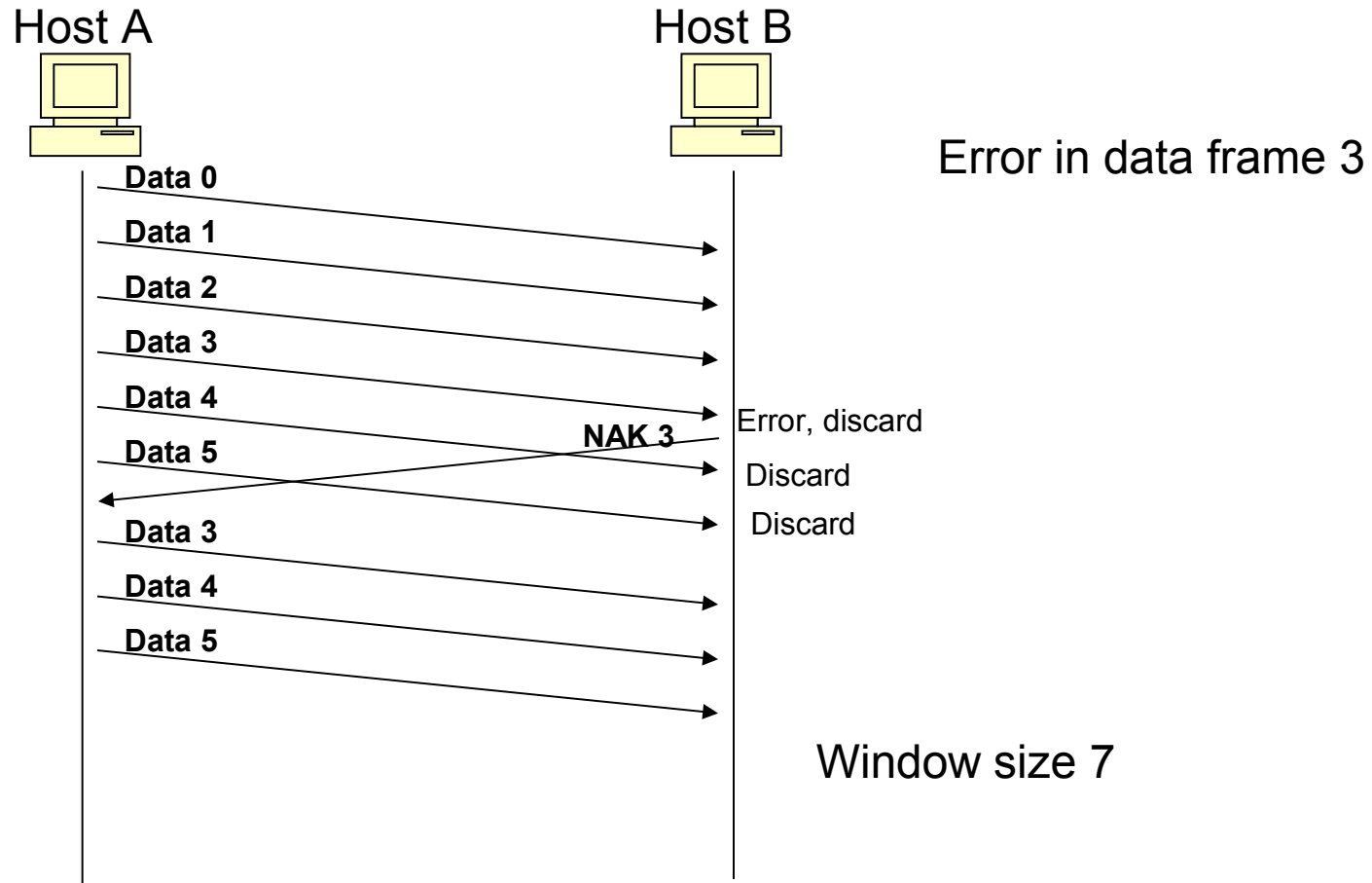
- Receiver sends positive ACK if a frame arrived safe and in order.
- If the frames are damaged/out of order, receiver is silent and discard all subsequent frames until it receives the one it is expecting.
- The silence of the receiver causes the timer of the unacknowledged frame to expire.
- Then the sender resends all frames, beginning with the one with the expired timer.
- For example, suppose the sender has sent frame 6, but the timer for frame 3 expires (i.e. frame 3 has not been acknowledged), then the sender goes back and sends frames 3, 4, 5, 6 again. Thus it is called Go-Back-N-ARQ
- The receiver does not have to acknowledge each frame received, it can send one cumulative ACK for several

operation

- The sender keeps track of the outstanding frames and updates the variables and windows as the ACKs arrive.



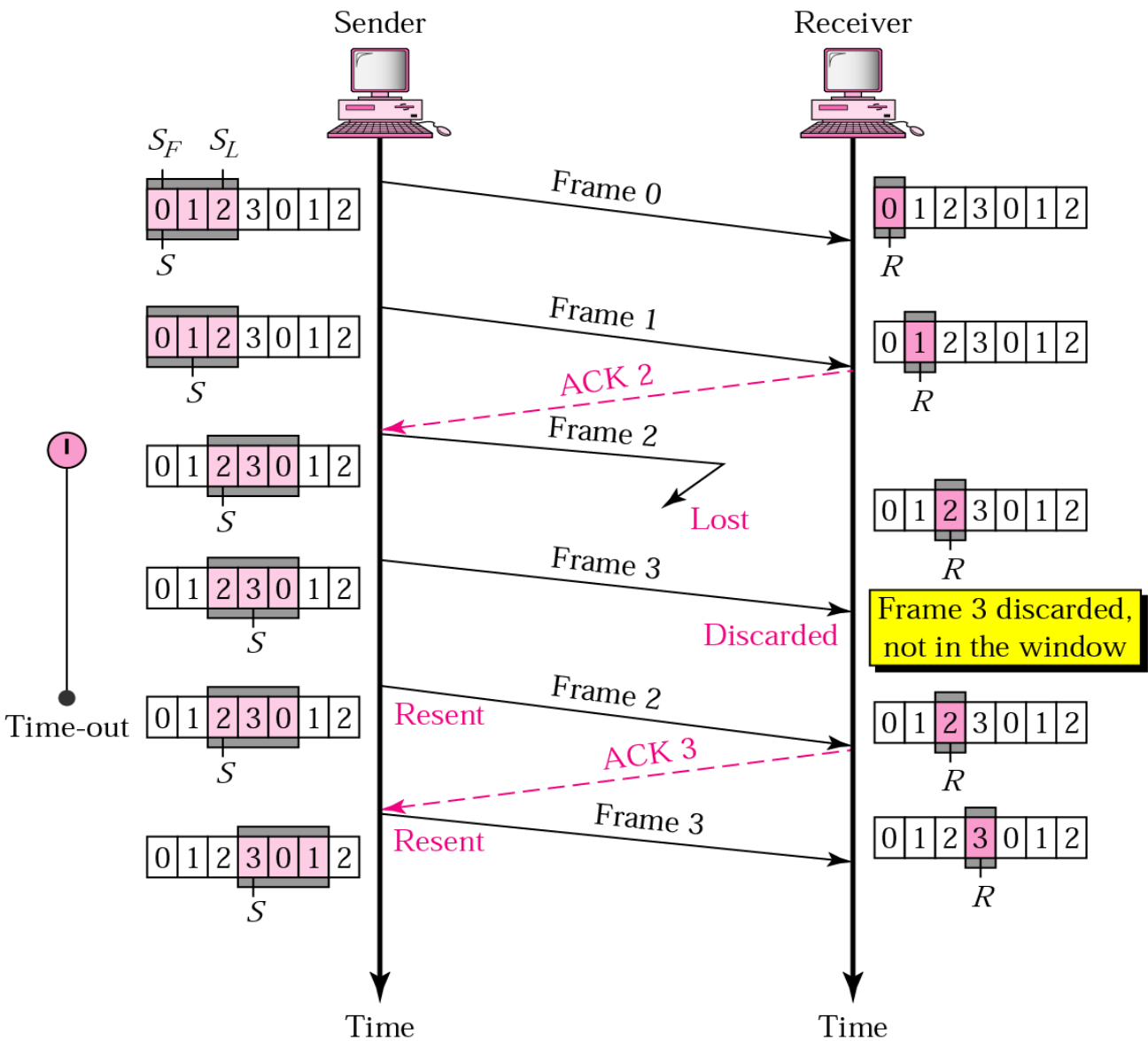
Go-Back-n ARQ



Go-Back-n ARQ

- Lost ACK:
 - When sender reaches window capacity, it starts a timer
 - If timer expires, it resends all outstanding (unACKed) frames
 - The receiver discards possible duplicate frames and sends another ACK

Go Back N(RQ), frame

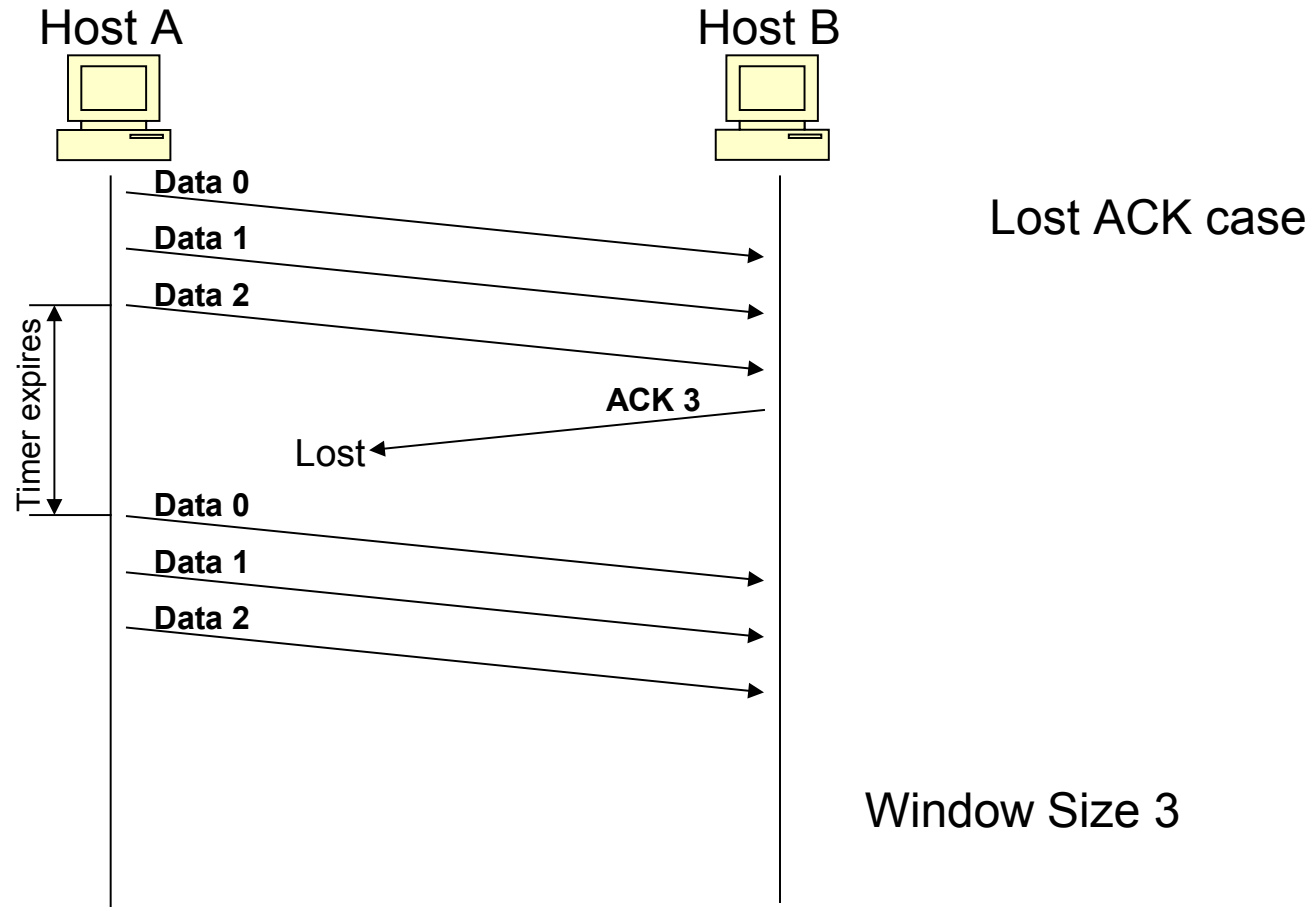


- Frame 2 is lost
- When the receiver receives frame 3, it discards frame 3 as it is expecting frame 2 (according to window).
- After the timer for frame 2 expires at the sender site, the sender sends frame 2 and 3. (go back to ?)

Go-Back-N ARQ, damaged/lost/delayed ACK

- If an ACK is damaged/lost, we can have two situations:
- If the next ACK arrives before the expiration of any timer, there is no need for retransmission of frames because ACKs are cumulative in this protocol.
- If ACK1, ACK2, and ACK3 are lost, ACK4 covers them if it arrives before the timer expires.
- If ACK4 arrives after time-out, the last frame and all the frames after that are resent.
- Receiver never resends an ACK.
- A delayed ACK also triggers the resending of frames

Go-Back-n ARQ



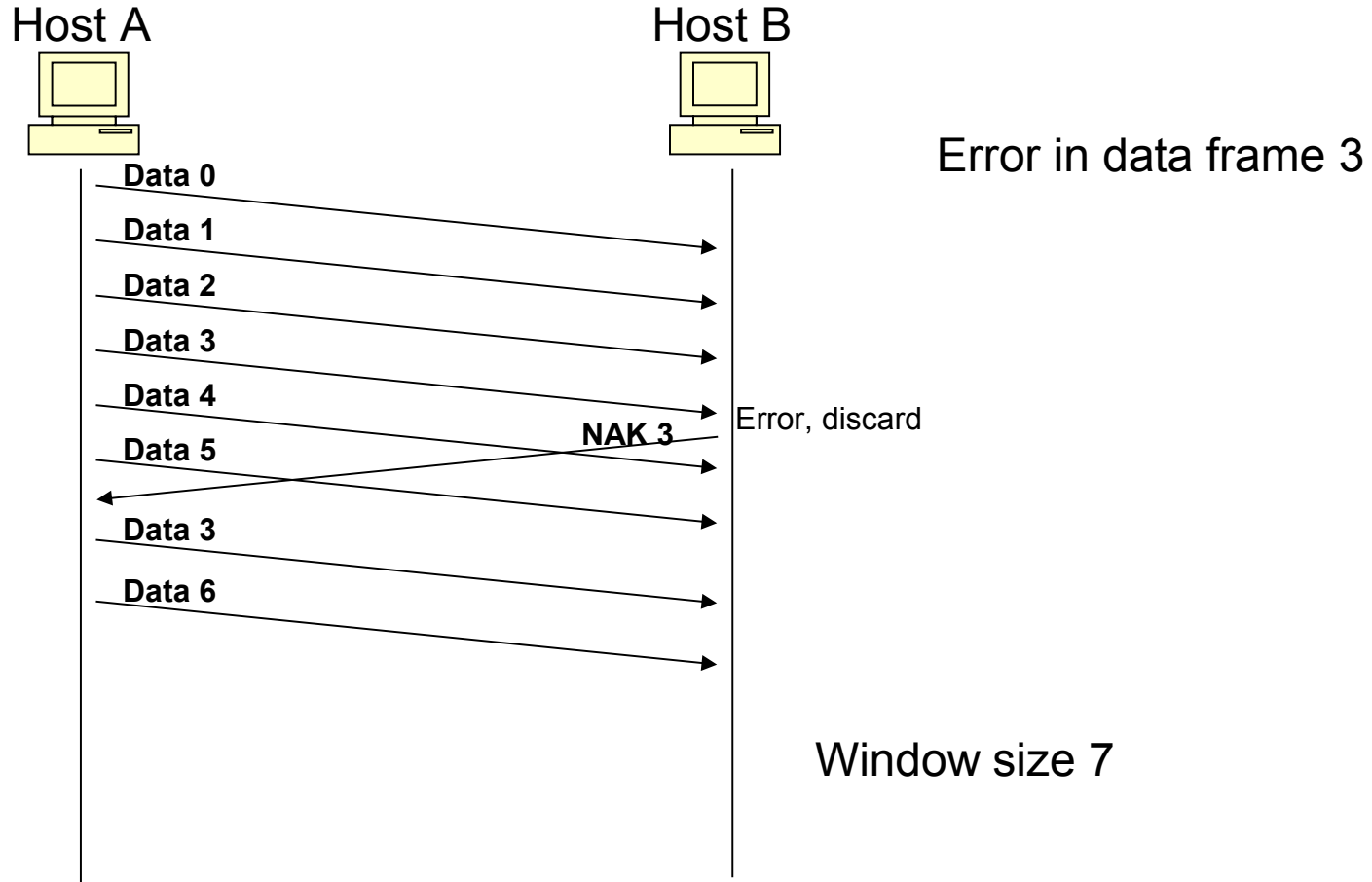
Selective Repeat ARQ

- Only damaged and lost frames are retransmitted
- ACK numbers refer to the last correctly received frame, not next frame expected!
- Damaged Frames:
 - After receiving a damaged frame, receiver sends a NAK and continues accepting other frames
 - Sender only retransmits the missing frame

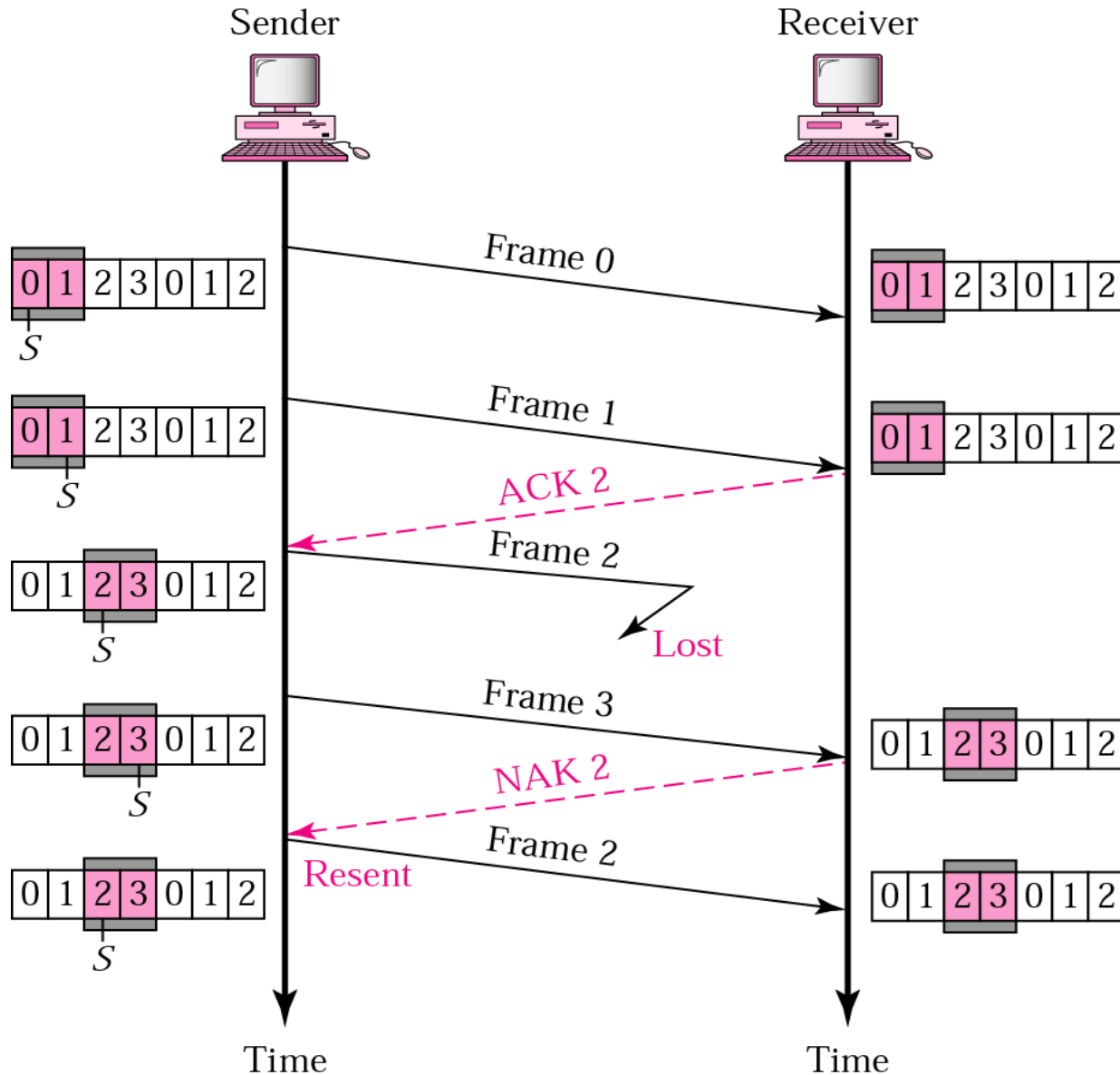
Selective Repeat ARQ, sender and receiver windows

- Go-Back-N ARQ simplifies the process at the receiver site. Receiver only keeps track of only one variable, and there is no need to buffer out-of-order frames, they are simply discarded.
- However, Go-Back-N ARQ protocol is inefficient for noisy link. It bandwidth inefficient and slows down the transmission.
- In Selective Repeat ARQ, only the damaged frame is resent. More bandwidth efficient but more complex processing at receiver.
- It defines a negative ACK (NAK) to report the sequence number of a damaged frame before the timer expires.

Selective Repeat ARQ



Selective Repeat ARQ, lost frame



- Frames 0 and 1 are accepted when received because they are in the range specified by the receiver window. Same for frame 3.
- Receiver sends a NAK2 to show that frame 2 has not been received and then sender resends only frame 2 and it is accepted as it is in the

size

- Size of the sender and receiver windows must be at most one-half of 2^m . If $m = 2$, window size should be $2^m / 2 = 2$. Fig compares a window size of 2 with a window size of 3. Window size is 3 and all ACKs are lost, sender sends duplicate of frame 0, window of the receiver expect to receive frame 0 (part of the window), so accepts frame 0, as the 1st frame

