

---

# *Redes de Computadores*

## **The Data Link Layer**

*Manuel P. Ricardo*

*Faculdade de Engenharia da Universidade do Porto*

- 
- » *What are the main services and functions of the Data Link layer?*
  - » *What is a frame? How to frame data? Why is stuffing important?*
  - » *What is the relationship between Bit Error Ratio and Frame Error Ratio?*
  - » *How to detect errors in a frame?*
  - » *How does Cyclic Redundancy Check operate?*
  - » *What are the CRC error detection capabilities?*
  - » *What is the purpose of Automatic Repeat ReQuest (ARQ)?*
  - » *What are the common ARQ mechanisms?*
  - » *How does Stop & Wait ARQ work?*
  - » *How does Go Back N ARQ ARQ work?*
  - » *How does the Selective Reject ARQ work?*
  - » *Why are sequence numbers important in ARQ mechanisms?*
  - » *What is the efficiency of the ARQ mechanisms?*
  - » *What mechanisms are employed in Ethernet, PPP and WLAN?*
  - » *What are the differences between End-to-End ARQ and Link-by-Link ARQ?*
  - » *Where are the ARQ mechanisms used in the TCP/IP reference model ?*

---

## *Data Link layer functions and services*

# *Data Link Layer – Functions and Services*

---

## ◆ Main functions

- » Provide service to the network layer
- » Eliminate/reduce transmission errors
- » Regulate data flow

Slow receivers not swamped by fast senders

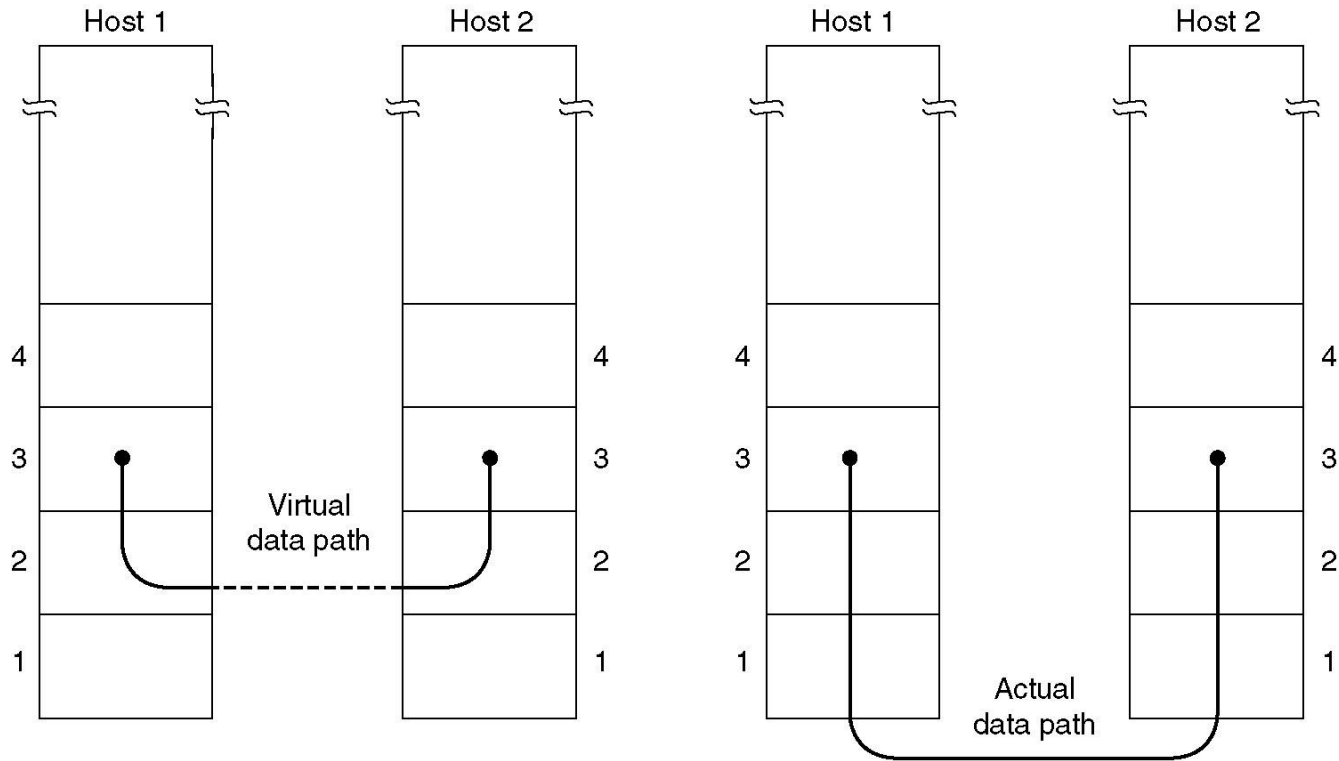
## ◆ Services provided

- » Unacknowledged connectionless service
- » Acknowledged connectionless service
- » Acknowledged connection-oriented service

5	Application layer
4	Transport layer
3	Network layer
2	Data Link layer
1	Physical layer

# *Services Provided to Network Layer*

---



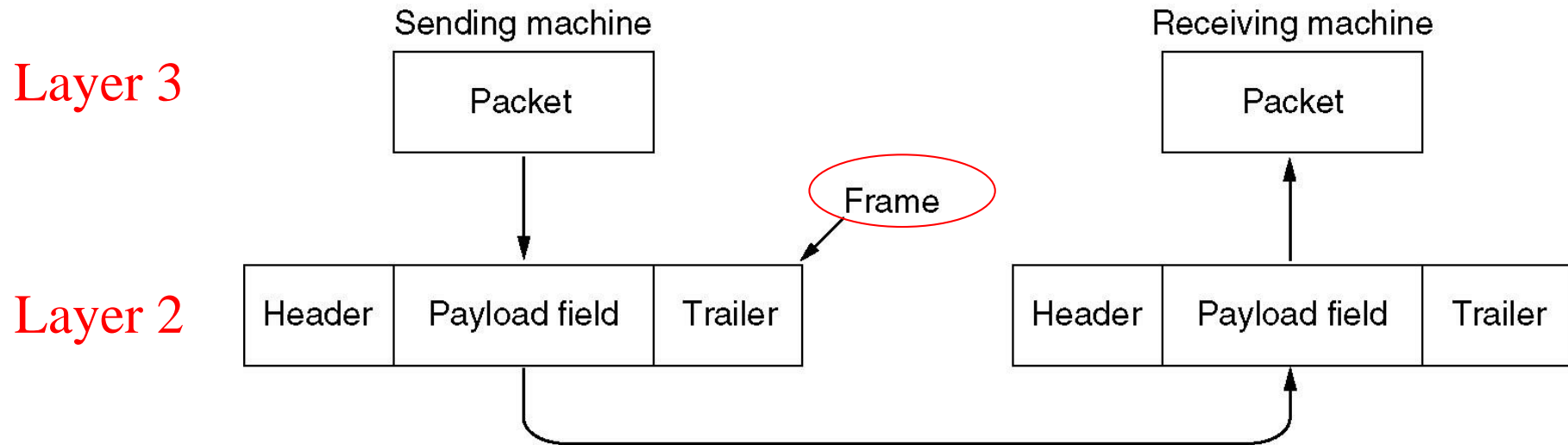
(a)

(a) Virtual communication

(b)

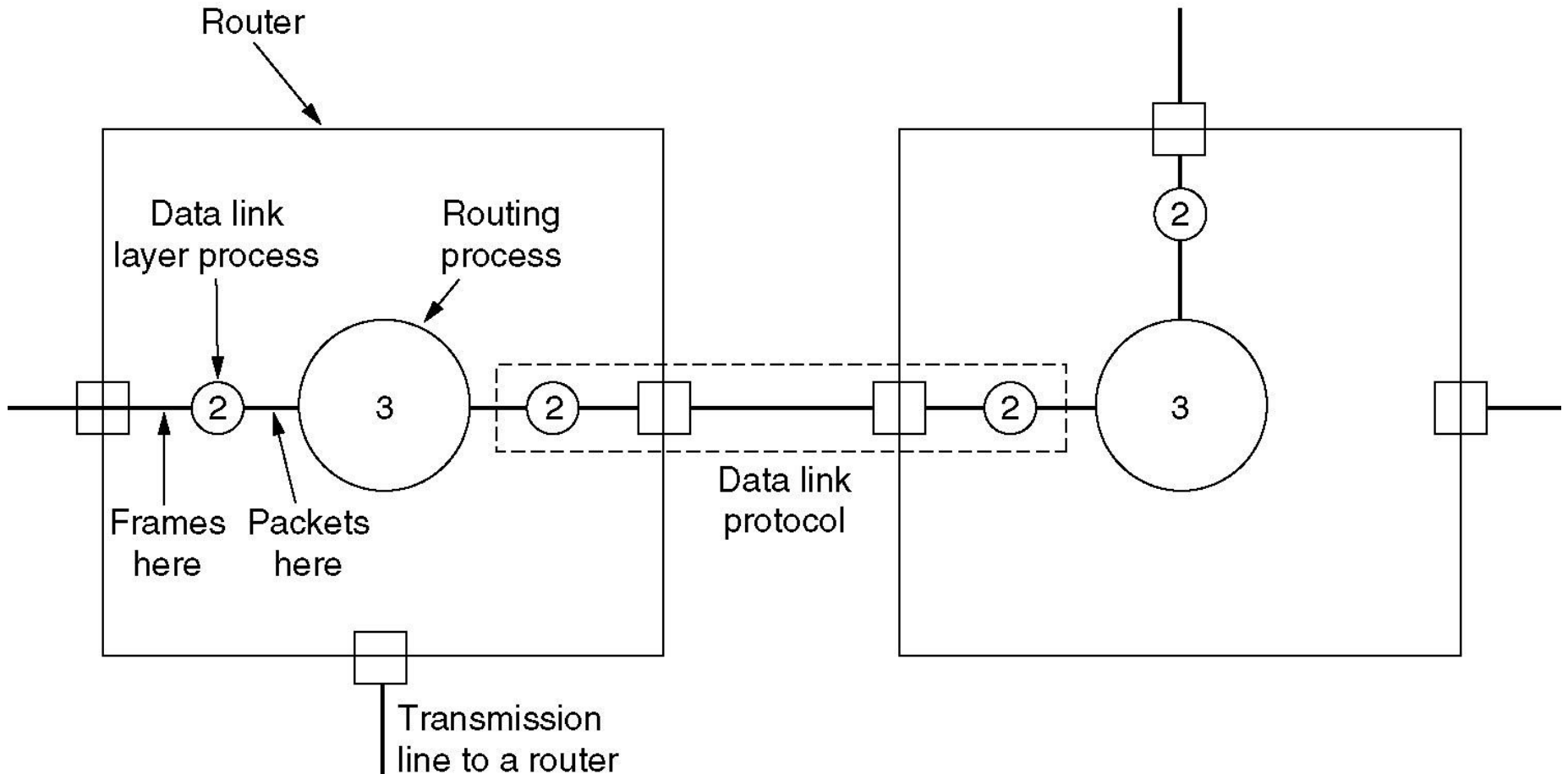
(b) Actual communication

# Layer 3 Packets and Layer 2 Frames



# *Placement of the Data Link Protocol*

---



---

*Framing*



## *To Think*

---

[Sender] → ..1001101101101101101101110101.. → [Receiver]

Where is the data? Where does the frame start and stop?

**How to split this bit stream into frames (sets of bits)?**

# *Framing*

---

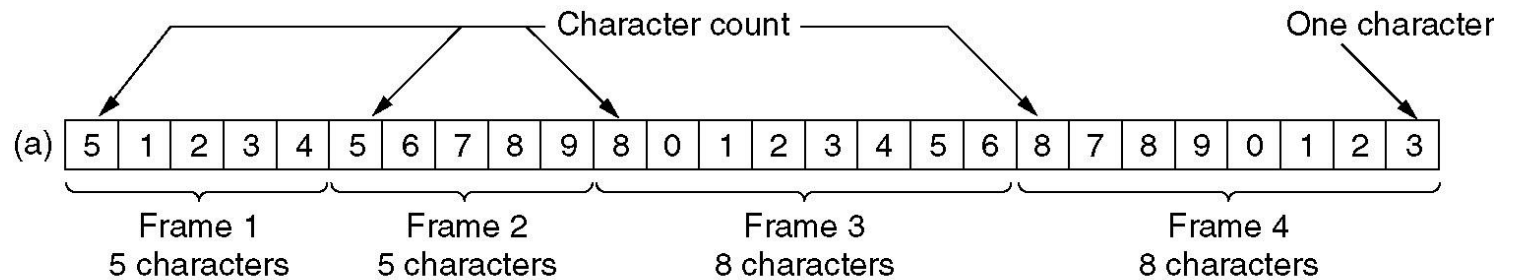
- ♦ [Sender] → ..1001101101101101101101101110101.. → [Receiver]
  - » Where is the data?
  - » Where does the frame start and stop?
  
- ♦ Three methods
  - » **Character count**
  - » **Flag bytes with byte stuffing**
  - » **Start and ending flags, with bit stuffing**

# *Framing – Character count*

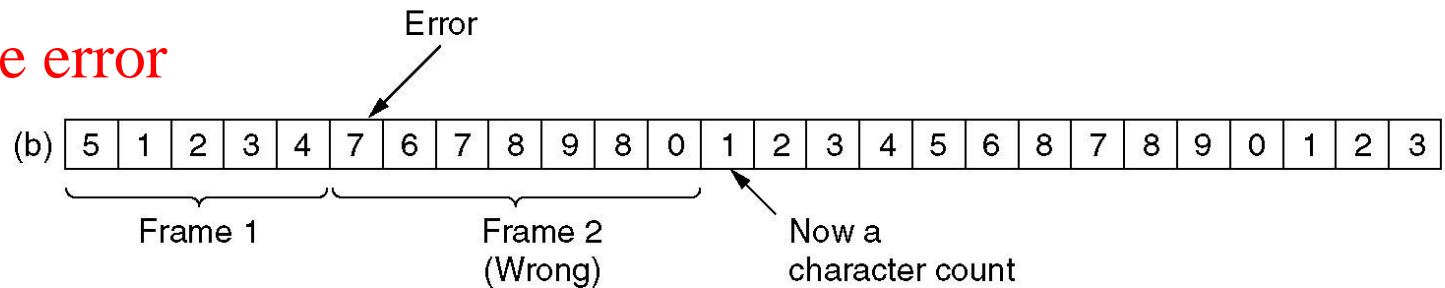
---

A stream of characters

(a) **Without errors**



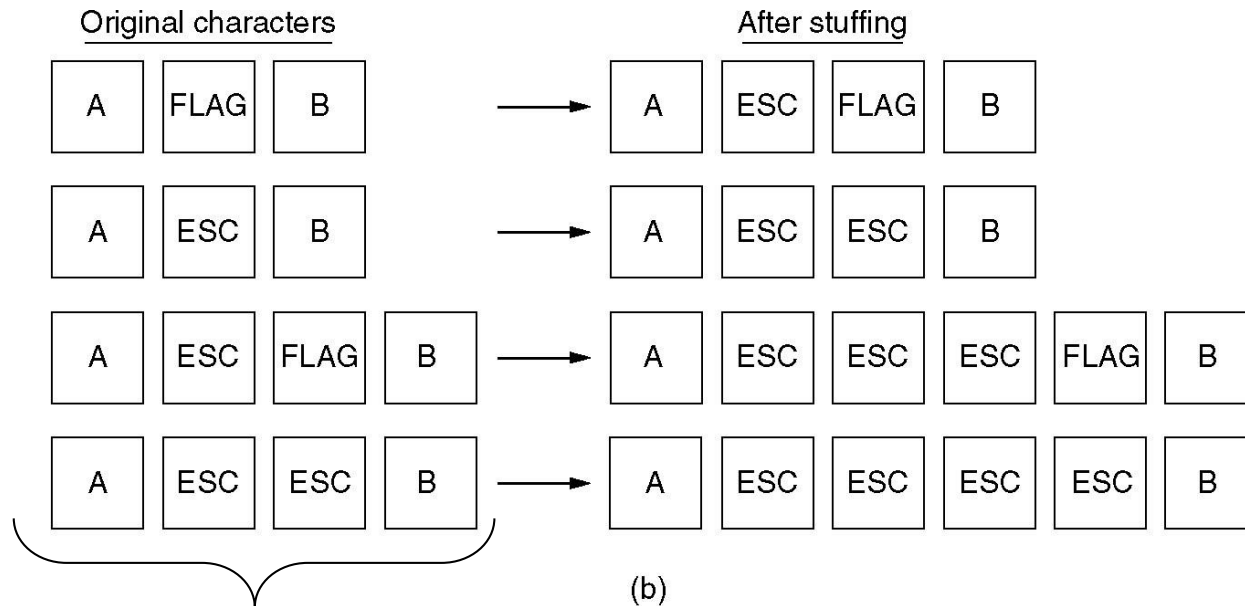
(b) **With one error**



# Framing - Flag bytes with *byte stuffing*



(a)



(b)

Problem if frame  
has internal  
character equal to  
FLAG!

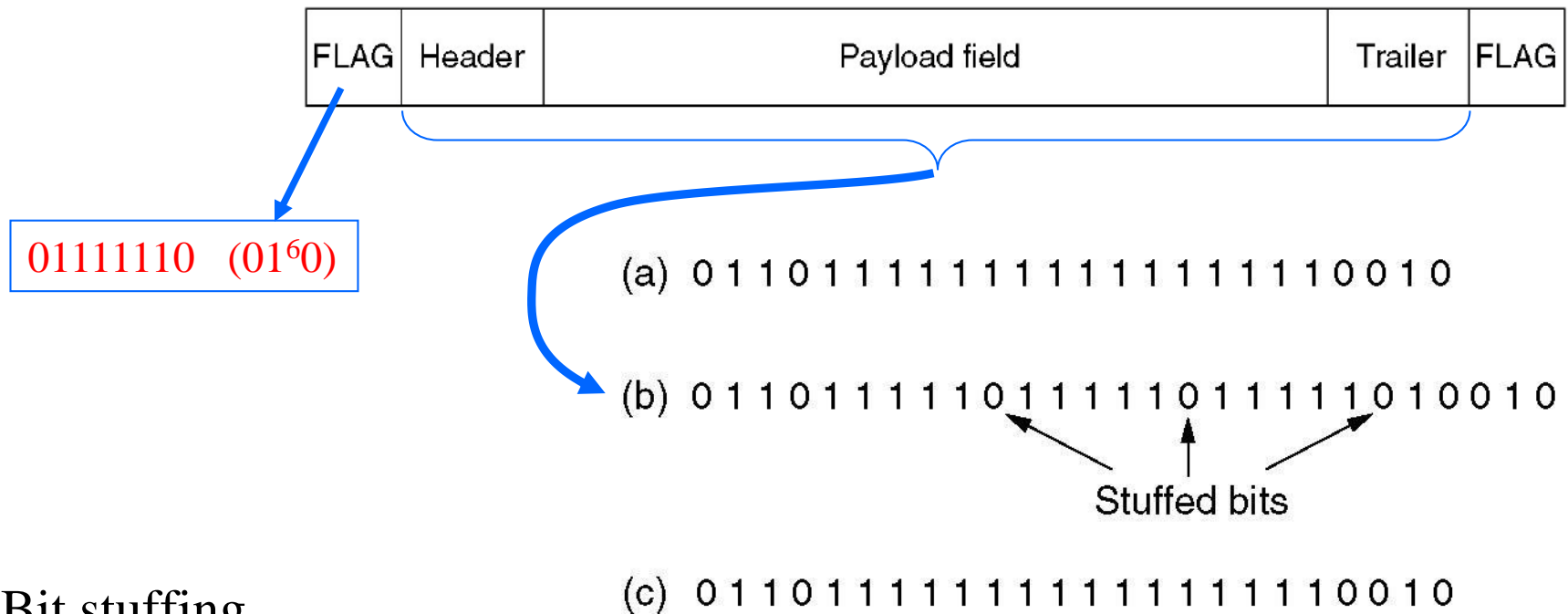
Problem solved  
by stuffing  
mechanism!

*Data to be transported in the payload field*

(a) A frame delimited by flag bytes

(b) Four examples of byte sequences before and after stuffing

## *Framing - Start and ending flags, with **bit stuffing***



### Bit stuffing

(a) The original data

(b) The data as it appears on the line:  $1^5 \rightarrow 1^5\underline{0}$

(c) The data as stored in receiver's memory **after destuffing**:  $01^5\underline{0} \rightarrow 01^5$

---

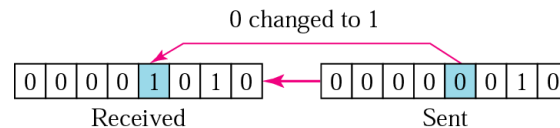
*Error detection*

# *Types of Errors*

---

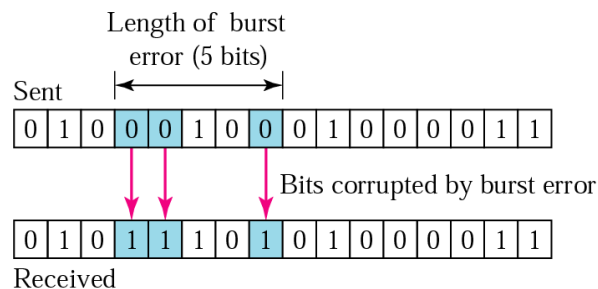
## ◆ Simple Error

- » Random and independent from previous error



## ◆ Errors in burst

- » Not independent; affect neighbour bits
- » Burst length defined by the first and last bits in error



# *To Think*

---

- ◆ Assume

- $p$  – bit error probability (or **Bit Error Ratio – BER**)
- $n$  – frame length
- Independent errors
- FER: Frame Error Ratio

- ◆ Student A explains to Student B

why  $P[\text{frame has no errors}] = (1 - p)^n$

- ◆ Student B explains to Student A

why  $P[\text{frame has errors}] = 1 - (1 - p)^n \Leftrightarrow FER = 1 - (1 - BER)^n$



# Counting Errors

---

## ◆ Assume

- $p$  – bit error probability (or **Bit Error Ratio – BER**)
- $n$  – frame length
- Independent errors

◆  $P[\text{frame has no errors}] = (1 - p)^n$

the  $n$  bits are good!

◆  $P[\text{frame has errors}] = 1 - (1 - p)^n$

$P[\text{frame has errors}] = \text{Frame Error Ratio (FER)}$

$$FER = 1 - (1 - BER)^n$$

$p = 10^{-7}$  (**good wired channel**)

$n = 10^4$  (~ Ethernet frame length)

$$FER = 1 - (1 - 10^{-7})^{10^4} \approx 10^{-3}$$

◆  $P[1 \text{ bit received in error}] = \binom{n}{1} p(1 - p)^{n-1}$

◆  $P[i \text{ bits received in error}] = \binom{n}{i} p^i (1 - p)^{n-i}$

$p = 10^{-3}$  (**wireless channel**)

$n = 10^4$  (~ Ethernet frame length)

$$FER = 1 - (1 - 10^{-3})^{10^4} \approx 1$$

# *Error Techniques*

---

- ◆ Error techniques required!
  - » Detection (and correction)
  
- ◆ Effectiveness of **error detection** technique (code) characterized by
  - » Minimum distance of code: **d**
    - min number of bit errors **undetected** in a block of  $n$  bits
    - if fewer than  $d$  errors occur, errors are detected
  
  - » Burst detecting ability: **B**
    - max burst length of errors **detected**

# *Error Detection Techniques*

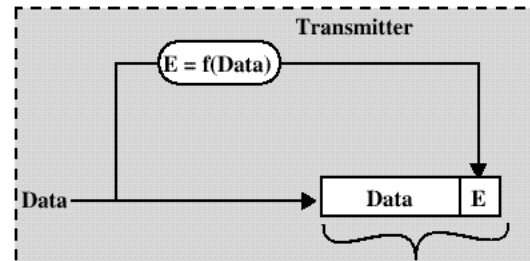
---

- ◆ Used by the receiver to determine if a packet contains errors
  - » If a packet is found to contain errors, the receiver may request the transmitter to re-send the packet

- ◆ Introducing redundancy →

»  $k \rightarrow k+r$

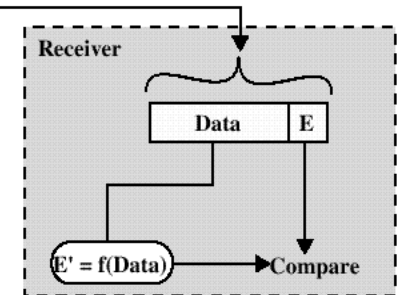
k: data bits; r: redundancy bits



$E, E'$  = error detecting codes  
 $f$  = error detecting code function

- ◆ Error detection techniques

- » Parity check
- » Cyclic Redundancy Check (CRC)
- » ...



# Simple Parity Check

---

- ♦ One parity bit added to every  $k$  information bits so that

- » The total number of bits 1 even → even parity

1110111	1101110	1010110	1101100	1100100
11101110	11011101	10101100	11011000	11001001

- » The total number of bit 1 is odd → odd parity

- ♦ Detection of

- » simple errors

- » any number of odd errors in a block of  $k+1$  bits

- ♦ Undetected

- » Even number of errors in a block

$n=k+1$ , block size

$p$ : bit error probability

$$P(\text{undetected}) = \sum_{i \text{ even}} \binom{n}{i} p^i (1-p)^{n-i}$$

- ♦ Used in Character Oriented protocols

# *Bi-dimensional Parity*

---

- ◆ Blocks represented in rows
  - » Parity bit per row; parity bit per column

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1
<hr/>							0
1	0	1	1	1	1	1	0

Vertical checks

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1
<hr/>							0
1	0	1	1	1	1	1	0

Horizontal checks

- ◆ Minimum code distance  $d=4$ 
  - Any four errors in a rectangular configuration becomes undetectable

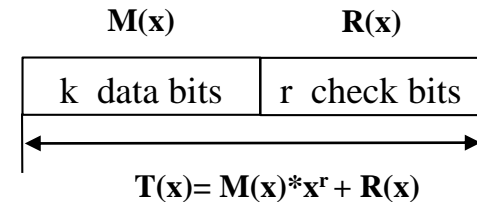
# Cyclic Redundancy Check (CRC)

---

- ♦ Bit string represented as a polynomial
  - »  $110011 \rightarrow x^5 + x^4 + x + 1$
- ♦ Module 2 operations
  - » Additions and subtractions identical to **exclusive OR**
  - » **no carry, no borrow**
- ♦  $M(x)$ ;  $R(x)$ ;  $T(x) = M(x) * x^r + R(x)$
- ♦ How to compute the check bits:  $R(x)$ ?
  - » Choose a generator string  $G(x)$  of length  $r+1$  bits
  - » Choose  $R(x)$  such that  **$T(x)$  is a multiple of  $G(x)$** :  $T(x) = A * G(x)$
- ♦  $T(x) = M(x)x^r + R(x) = A * G(x) \iff$ 

$$M(x)x^r = A * G(x) + R(x) \quad (\text{mod } 2 \text{ arithmetic})$$

$$\Rightarrow R(x) = \text{remainder of } M(x)x^r / G(x)$$
- ♦ Choice of  $G(x)$  is very important! (  **$G(x) = x^r + \dots + 1$**  )



# CRC - Generating $R(x)$

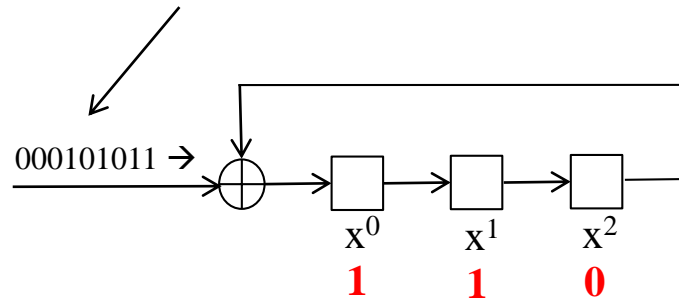
- ♦  $r=3, x^r=x^3$  ;  $G(x)=x^3+1$  (1001)
- ♦  $M(x)=x^5+x^4+x^2+1$  (110101)
- ♦  $M(x) * x^3 = x^8+x^7+x^5+x^3$  (110101000)
- ♦  $R(x)$  = remainder of  $M(x)x^r / G(x)$
- ♦  $R(x)=x+1$  (011)
- ♦ Sent word  
 »  $T(x)=M(x) * x^r + R(x) = x^8+x^7+x^5+x^3+x+1 = 110101011$

$M(x) * x^3$								$G(x)$			
1	1	0	1	0	1	0	0	1	0	0	1
1	0	0	1	↓	↓	↓	↓	1	1	0	0
0	1	0	0	0	↓	↓	↓				
	1	0	0	1	↓	↓	↓				
	0	0	0	1	1	↓	↓				
		0	0	0	0	↓	↓				
		0	0	1	1	0	↓				
			0	0	0	0	↓				
			0	1	1	0	0	↓			
				1	0	0	1	↓			
				0	1	0	1	0			
					1	0	0	1			
					0	0	1	1			

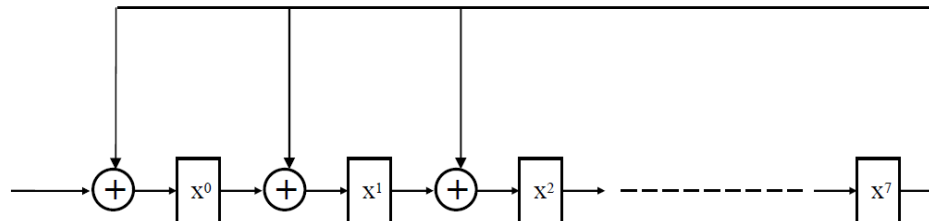
**R(x)**

# CRC - Generating $R(x)$ with a Shift Register

- ♦  $R(x)$  easily generated in hardware
- ♦  $G(x)=x^3+1$ 
  - »  $M(x) * x^3 = x^8 + x^7 + x^5 + x^3$  (110101000)
  - »  $R(x)=x+1$  (011)



- ♦  $G(x)=x^8+x^2+x+1$





# CRC – Checking at the Receiver

- ◆ Let  $T'(x)$  be the received word
  - »  $T'(x) = x^8 + x^7 + x^5 + x^3 + x + 1$  (110101 011)
- ◆ Divide  $T'(x)$  by  $G(x)$ 
  - » If remainder  $R(x) = 0 \rightarrow$  no errors
  - » If remainder  $R(x) \neq 0 \rightarrow$  errors have occurred

$T'(x)$										$G(x)$					
1	1	0	1	0	1	0	1	1		1	0	0	1		
1	0	0	1							1	1	0	0	1	1
0	1	0	0	0											
	1	0	0	1											
	0	0	0	1	1										
		0	0	0	0										
		0	0	1	1	0									
			0	0	0	0									
			0	1	1	0	1								
				1	0	0	1								
				0	1	0	0	1							
					1	0	0	1							
						0	0	0							
							0	0	0						

# *CRC - Performance*

---

- ◆ For  $r$  check bits per frame the following can be detected
  - » All patterns of 1, 2, or 3 errors ( $d > 3$ )
  - » All bursts of errors of  $r$  or fewer bits
  - » All errors consisting of an odd number of inverted bits
  
- ◆ Common polynomials
  - » ITU-16:  $r=16$ ,  $G(x) = x^{16} + x^{12} + x^5 + 1$  (1000100000100001)
  - » ITU-32:  $r=32$ ,  
 $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

---

*Automatic Repeat reQuest (ARQ)*

# *Automatic Repeat ReQuest (ARQ)*

---

- ◆ When the receiver detects errors in a frame  
how to ask the sender to retransmit the frame?
- ◆ ARQ mechanisms  
Mechanisms that automatically request the retransmission of
  - missing packets
  - packets with errors
- ◆ Three common ARQ schemes
  - » **Stop and Wait**
  - » **Go Back N**
  - » **Selective Repeat**

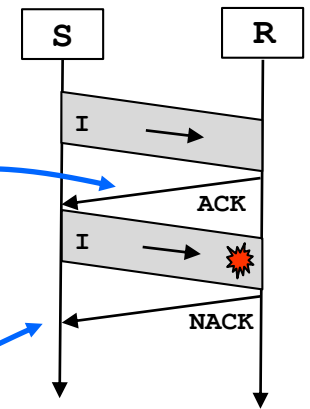
# Stop and Wait ARQ

## ◆ Sender

- » transmits Information frame **I**
- waits for positive confirmation **ACK** from receiver

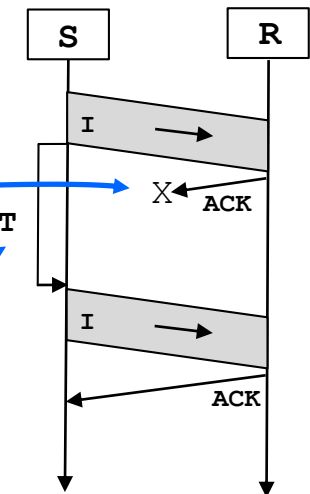
## ◆ Receiver: receives **I** frame

- » If **I** frame has no error → confirms with **ACK**
- » If **I** frame has error → sends **NACK**



## ◆ Sender

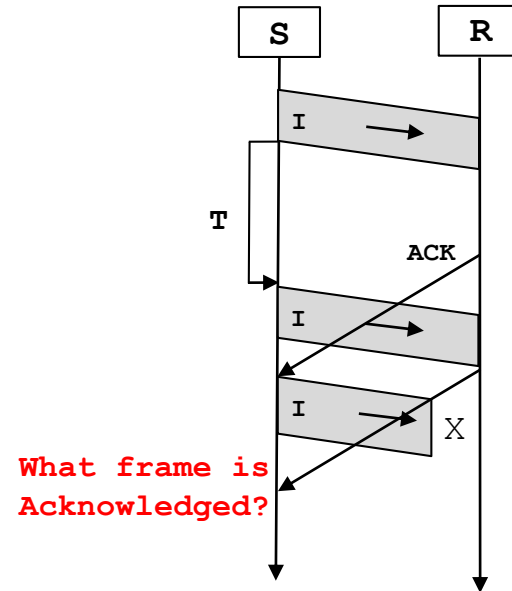
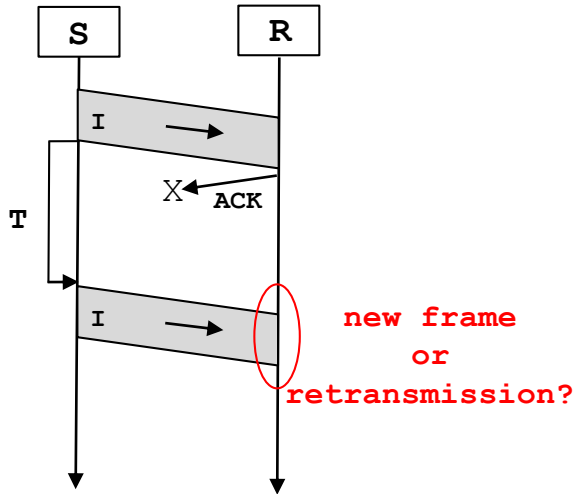
- » Receives **ACK** → proceeds and transmits **new** frame
- » Receives **NACK** → **retransmits** frame **I**



## ◆ Problem

- » What happens if **I**, **ACK** or **NACK** is lost?
- Timeout required!

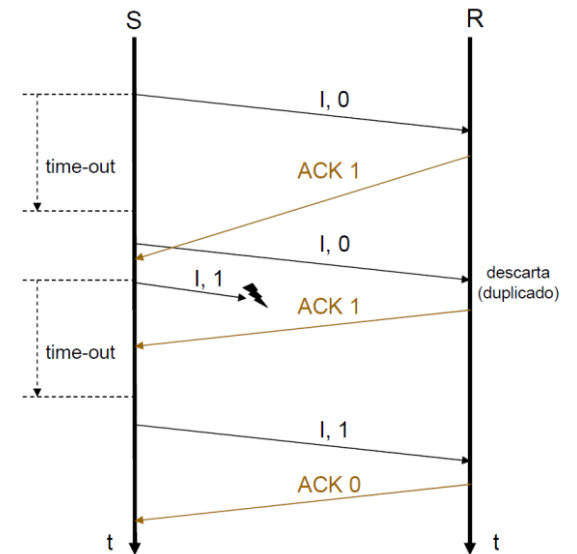
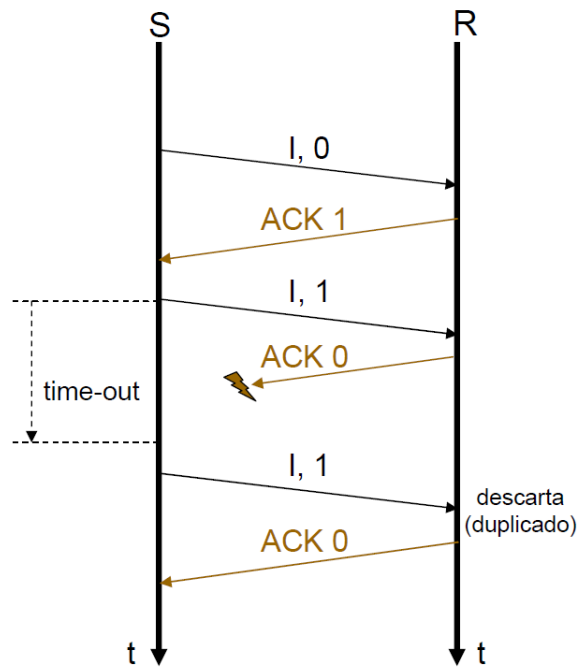
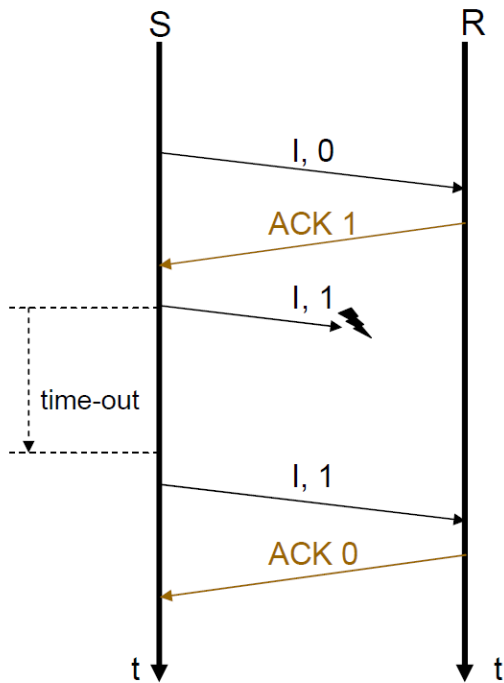
## *Stop and Wait ARQ – Sequence Numbers Required*



## ◆ Solution

- » **I** frames numbered: **I(0)**, **I(1)**
- » **ACK** frames numbered: **ACK(0)**, **ACK(1)**
- » **ACK(i)** indicates that receiver is waiting for frame **I(i)**
- » No **NACK** required
- » Module 2 numbers

# Stop and Wait ARQ – Examples



# Stop and Wait – Efficiency Example

## » WAN ATM

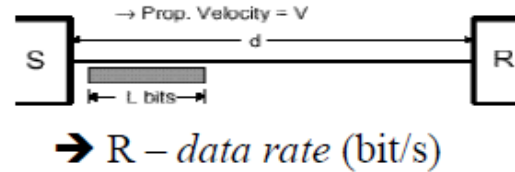
- $d = 1000 \text{ km}$
- $L = 424 \text{ bit}$ ,  $R = 155.52 \text{ Mbit/s}$
- $T_t = 2.7 \mu\text{s}$
- Fibra óptica  $\rightarrow 5 \mu\text{s/km} \rightarrow \tau = 5 \text{ ms}$
- $a = 1852$
- $S = 1 / 3705 = 0.0003$

## » LAN

- $d = 0.1 \sim 10 \text{ km}$
- $L = 1000 \text{ bit}$ ,  $R = 10 \text{ Mbit/s}$
- $T_t = 100 \mu\text{s}$
- Cabo coaxial  $\rightarrow 4 \mu\text{s/km} \rightarrow \tau = 0.4 \sim 40 \mu\text{s}$
- $a = 0.004 \sim 0.4$
- $S = 0.55 \sim 0.99$  (e se  $R = 100 \text{ Mbit/s?}$ )

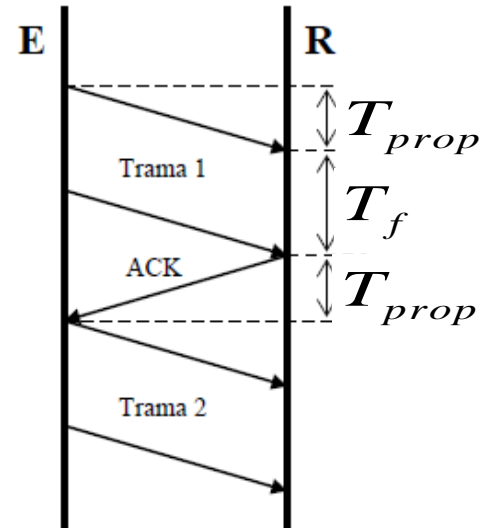
## » Modem sobre linha telefónica

- $d = 1000 \text{ m}$
- $L = 1000 \text{ bit}$ ,  $R = 28.8 \text{ kbit/s}$
- $T_t = 34.7 \text{ ms}$
- UTP  $\rightarrow 5 \mu\text{s/km} \rightarrow \tau = 5 \mu\text{s}$
- $a = 1.44 \cdot 10^{-4}$
- $S \simeq 1.0$



$$T_f = \frac{L}{R} = T_t$$

$$T_{prop} = \frac{d}{V} = \tau$$

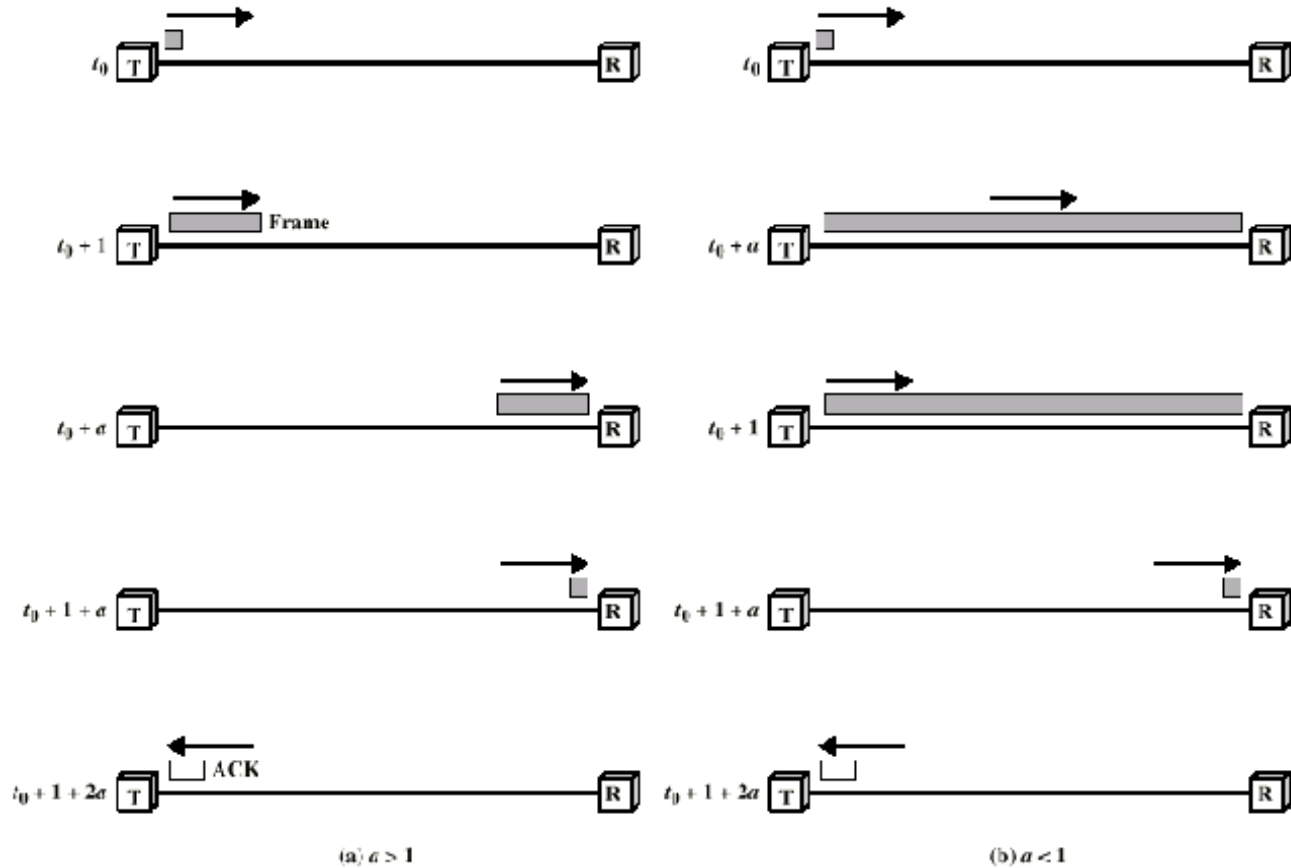


$$a = \frac{T_{prop}}{T_f}$$

$$S = \frac{T_f}{T_{prop} + T_f + T_{prop}} = \frac{1}{1 + 2a}$$



# Stop and Wait - Efficiency



Stop-and-Wait Link Utilization (transmission time = 1; propagation time =  $a$ )

# *Stop and Wait ARQ – Efficiency with Errors*

---

- »  $p_e$  – frame error probability
- »  $P[A=k]$ 
  - Probability of **k Attempts** required to transmit a frame with success

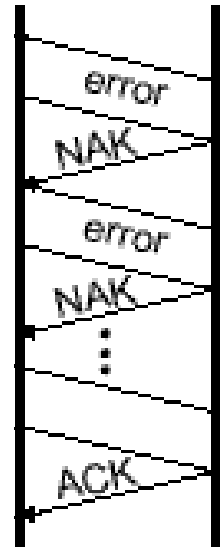
$$P[A = k] = p_e^{k-1} (1 - p_e)$$

- »  $E[A]$ 
  - expected number of Attempts to transmit a frame with success

$$E[A] = \sum_{k=1}^{+\infty} k * P[A = k] = \frac{1}{1 - p_e}$$

- » Efficiency

$$S = \frac{T_f}{E[A](T_f + 2T_{prop})} = \frac{1}{E[A](1 + 2a)} = \frac{1 - p_e}{1 + 2a}$$



## *To Think*

---

- ◆ Assume Sender and Receiver are separated by a large distance?  
How to improve the Efficiency of the Stop & Wait ARQ?

# *Go Back N ARQ (Sliding Window)*

---

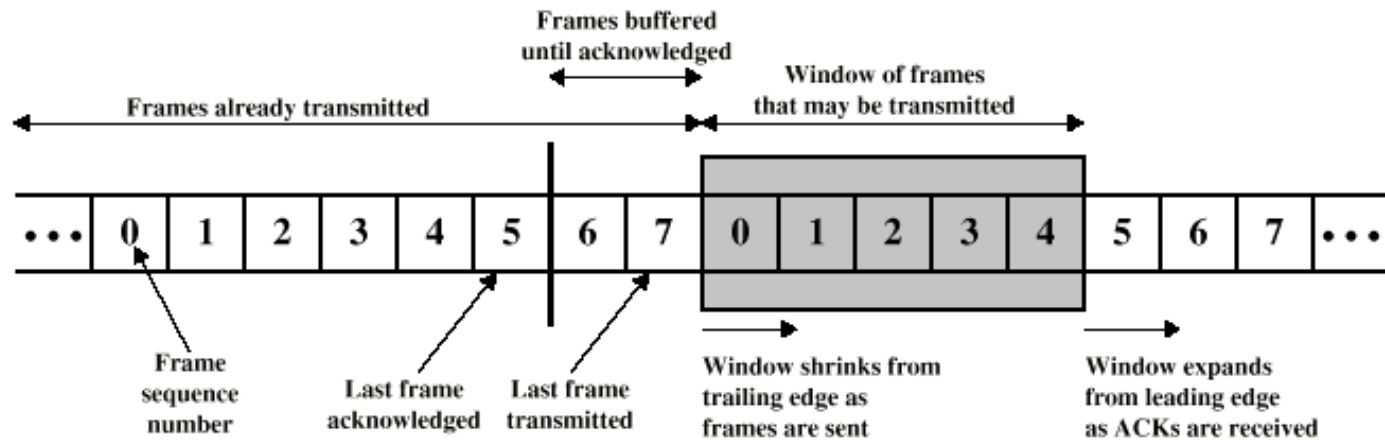
## ♦ Stop and Wait

- » inefficient when  $T_{\text{prop}} > T_f$  (  $a > 1$  )
- » sends only one frame per Round-Trip Time (  $\text{RTT} = 2 * T_{\text{prop}} + T_f$  )

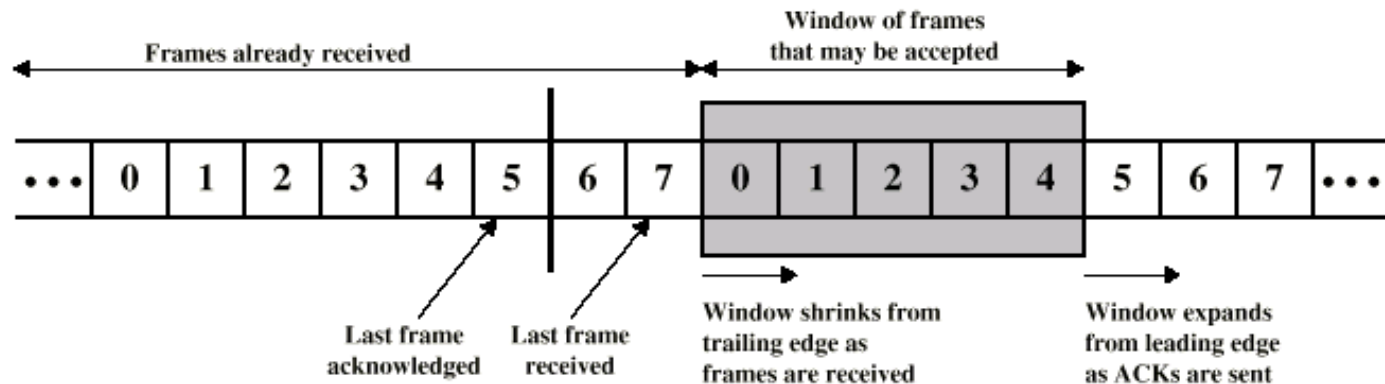
## ♦ Go Back N

- » allows transmission of new packets before earlier ones are acknowledged
- » uses a **Sliding Window** mechanism
  - sender can send packets that are within a “window” (range) of packets
  - window advances as acknowledgements for earlier packets are received

# Sliding Window - Model

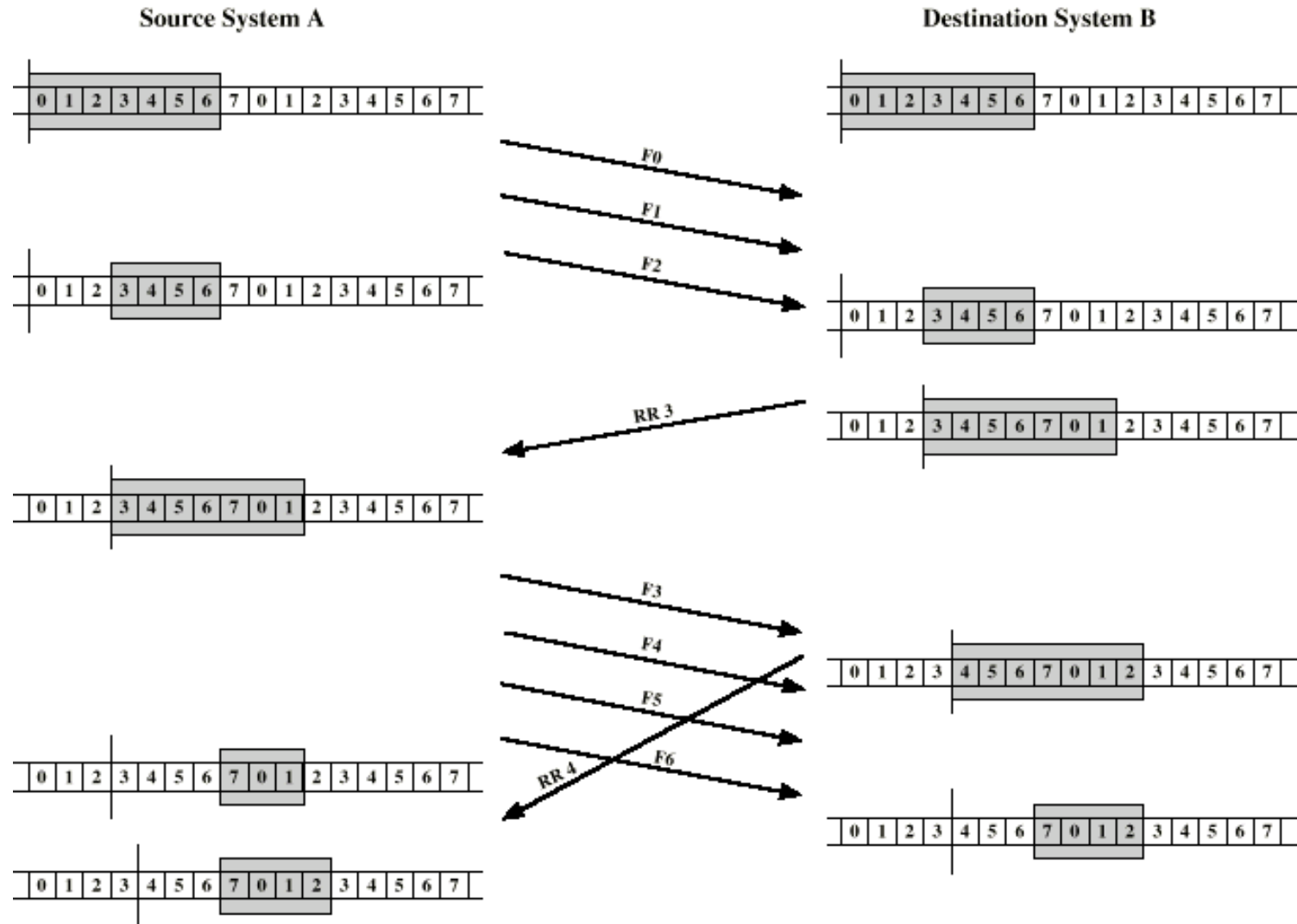


(a) Sender's perspective



(b) Receiver's perspective

# Sliding Window - Example



# *Go Back N ARQ – Basic Behaviour*

---

## ◆ Sender

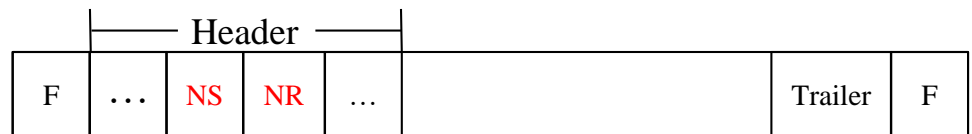
- » may transmit up to **W** frames without receiving RR  
RR - Receiver Ready = ACK
- » I frames are numbered sequentially I(NS): I(0), I(1), I(2), ...
- » Cannot send I(NS=i+W) until it has received the RR(NR=i)

## ◆ Receiver

- » does not accept frames out of sequence
- » sends RR(NR) to sender indicating
  - that **all the packets up to NR-1 have been received in sequence**
  - the sequence number, NR, of the next expected frame

## *Go Back N ARQ – Maximum Window, Extensions*

- ♦ Sequence numbers are represented module M
  - » NS, NR in  $\{0, 1, \dots, M-1\}$
- ♦ Maximum Window
  - »  $W = M-1 = 2^k - 1$
  - » k is number of bits used to code sequence numbers



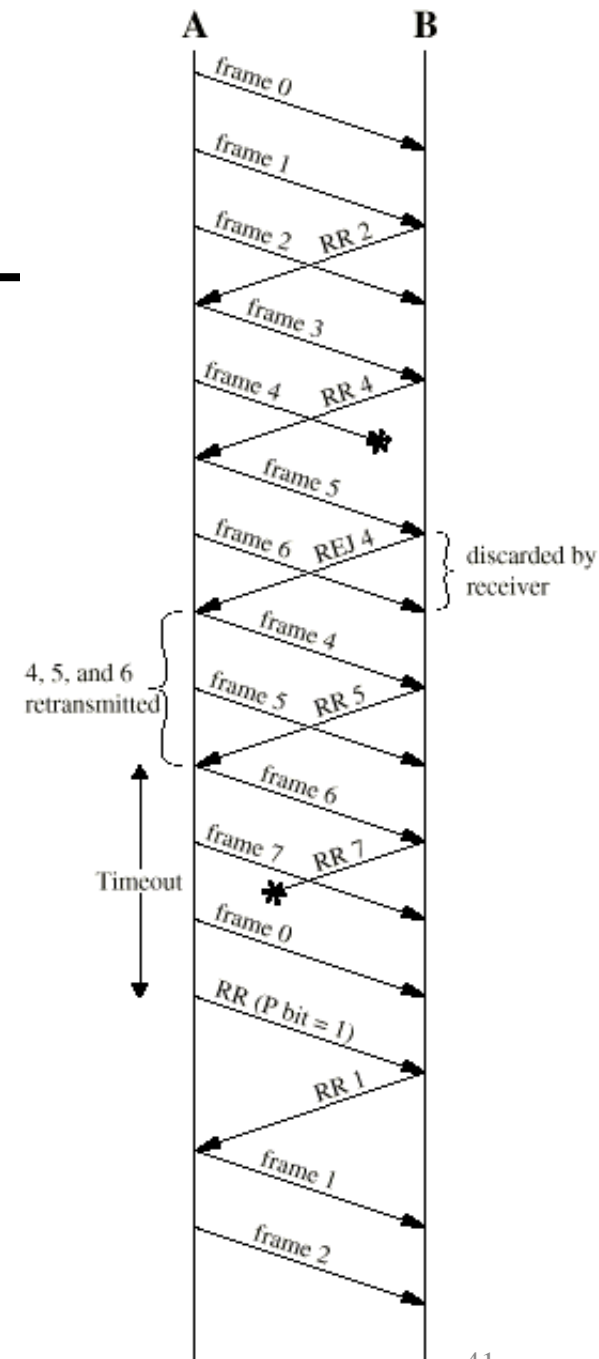
- ♦ Extensions to basic behaviour
  - » Piggybacking can be used for bidirectional flows
  - » RR information can be sent in the data packets of opposite direction



# *Go Back N ARQ – Behaviour under Errors*

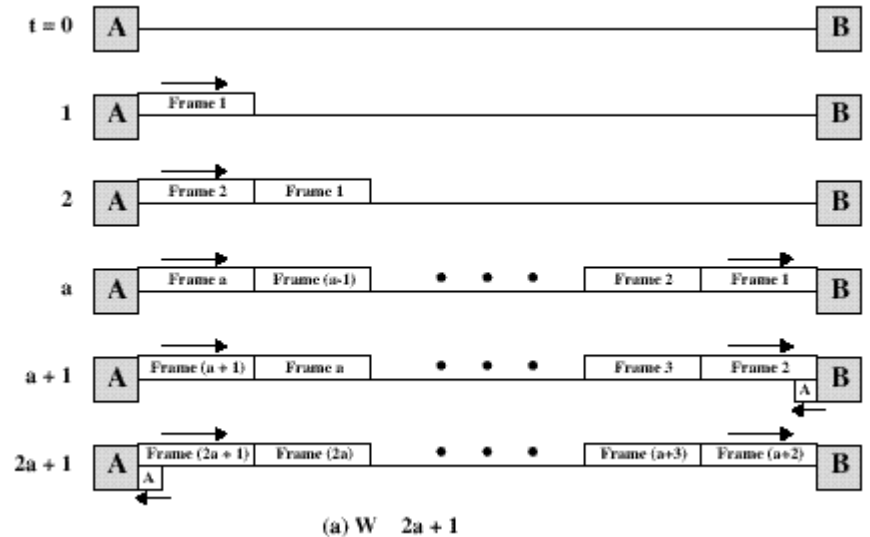
---

- ◆ Frame with errors  
is silently discard by the Receiver
- ◆ If Receiver receives Data frame out of sequence
  - » First out-of-sequence-frame?
    - Receiver sends REJ(NR)
    - NR indicates the next in-sequence frame expected
  - » Following out-of sequence-frames
    - Receiver discards them; no REJ sent
- ◆ When Sender receives REJ(NR=x), the Sender
  - » **Goes-Back** and retransmits I(x), I(x+1), ...
  - » Continues using Sliding Window mechanism
- ◆ If timeout occurs, the Sender
  - » requests the Receiver to send a RR message
  - » by sending a special message (RR command message)

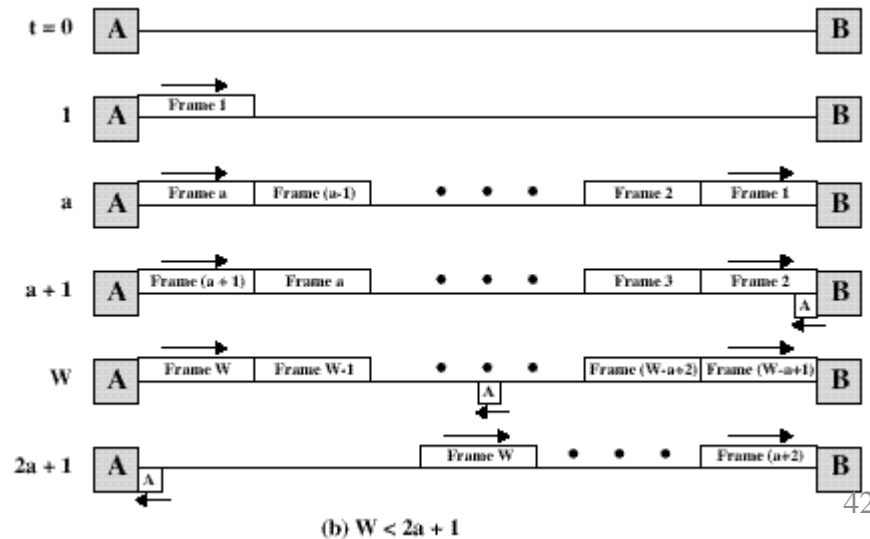


# Go Back N – Efficiency

♦ If  $W \geq 1+2a \rightarrow S = 1$

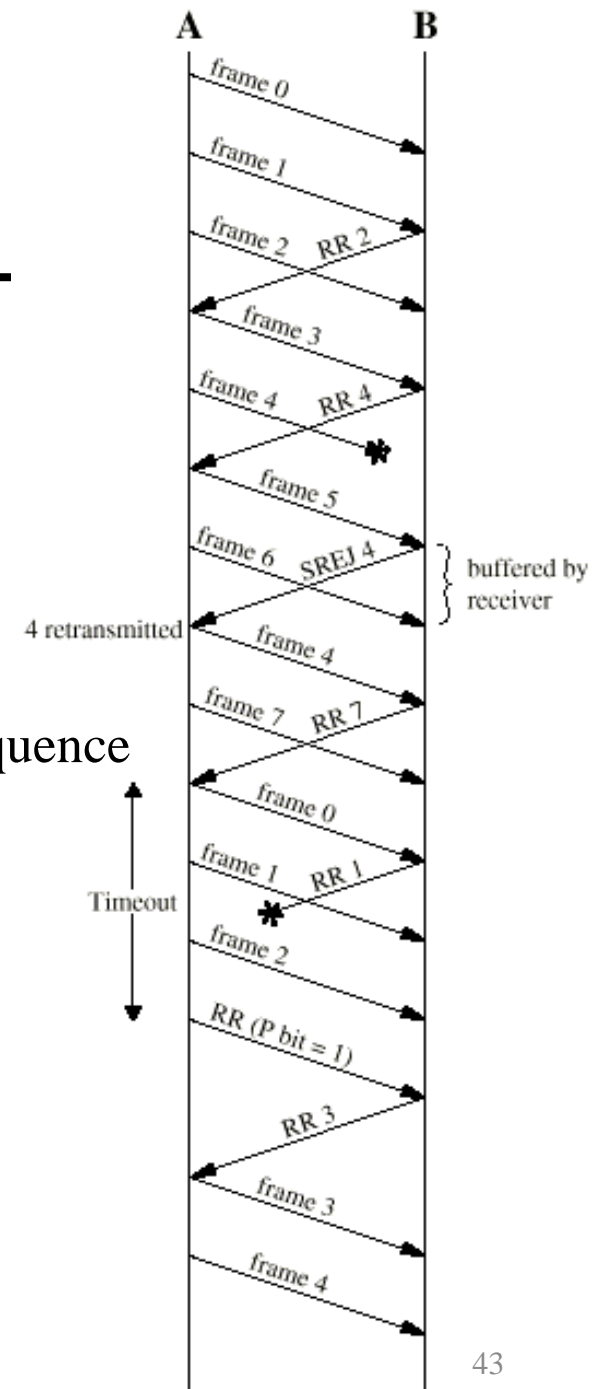


♦ If  $W < 1+2a \rightarrow S = W/(1+2a)$



# Selective Repeat ARQ

- ◆ Uses Sliding Window, but ...
- ◆ Receiver
  - » accepts out-of-sequence-frames
  - » confirms negatively, SREJ, a frame not arrived
  - » uses RR to confirm blocks of frames arrived in sequence
- ◆ Sender
  - » retransmits only the frames signaled by SREJ
- ◆ Adequate if  $W$  (a) is very large
- ◆ Maximum window size,  $W = \frac{M}{2} = 2^{k-1}$



# *Go-Back-N and Selective Repeat ARQ – Efficiency under Errors*

---

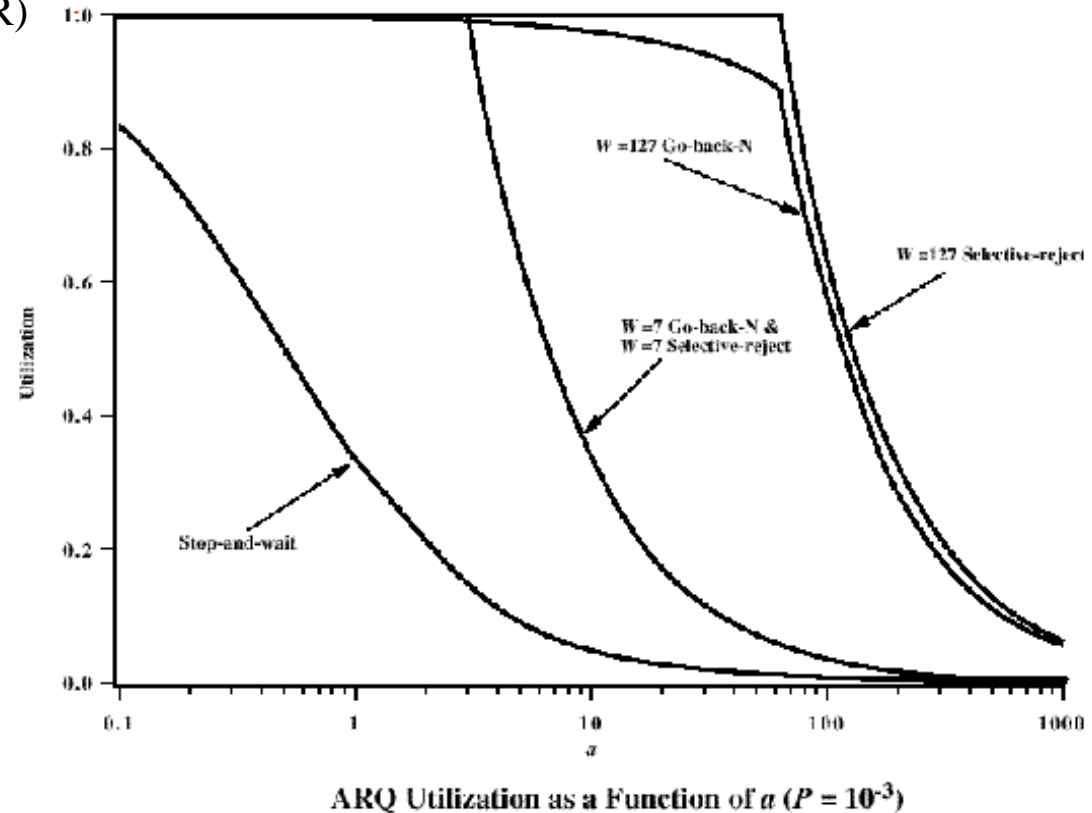
## ◆ *Go-Back-N ARQ*

$p_e$  – frame error probability (ratio, FER)

$$S = \begin{cases} \frac{1 - p_e}{1 + 2ap_e} & , W \geq 1 + 2a \\ \frac{W(1 - p_e)}{(1 + 2a)(1 - p_e + Wp_e)} & , W < 1 + 2a \end{cases}$$

## ◆ *Selective Repeat ARQ*

$$S = \begin{cases} 1 - p_e & , W \geq 1 + 2a \\ \frac{W(1 - p_e)}{1 + 2a} & , W < 1 + 2a \end{cases}$$



---

*Framing, Error detection and ARQ in common networks*

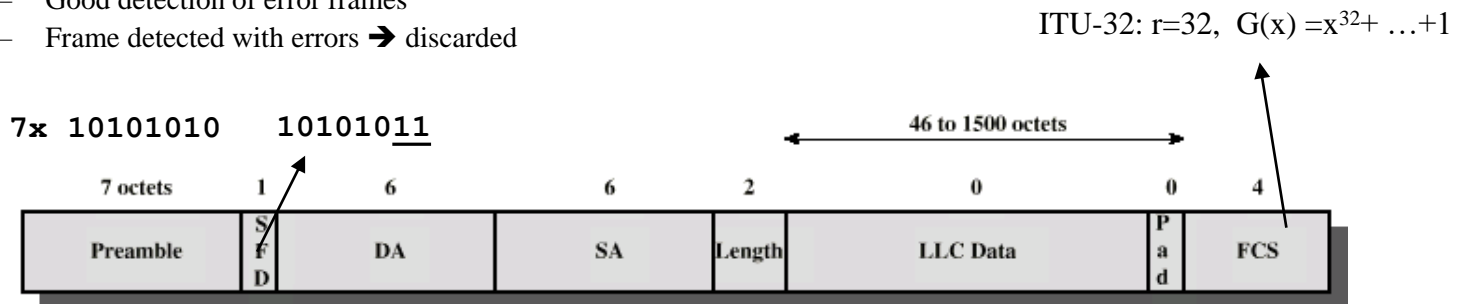
# Ethernet

- ♦ Framing
  - » Start of frame: preamble + SFD
  - » End of frame: end of signal transitions (Manchester code), length
- ♦ Error detection: FCS → ITU-32,  $G(x) = x^{32} + \dots + 1$
- ♦ No ARQ
  - » Bit Error ratio (BER) very low  
→ Frame Error Ratio (FER) low
  - » CRC/FCS strong
    - Good detection of error frames
    - Frame detected with errors → discarded

$p=10^{-7}$  (good wired channel)

$n=10^4$  (~ Ethernet frame length)

$P[\text{frame has errors}] = 1 - (1 - 10^{-7})^{10^4} \approx 10^{-3}$



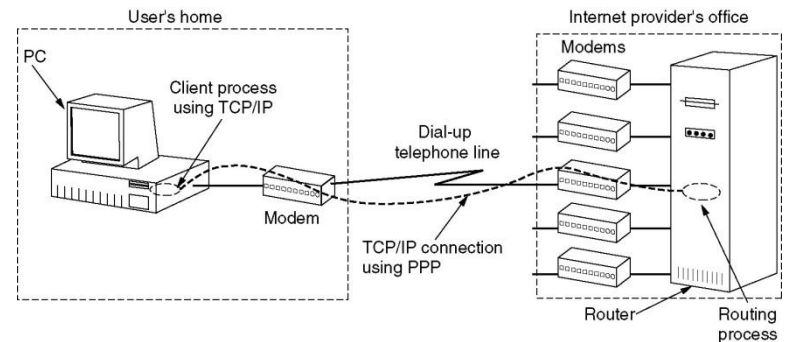
SFD = Start of frame delimiter  
 DA = Destination address  
 SA = Source address  
 FCS = Frame check sequence

# Point to Point Protocol

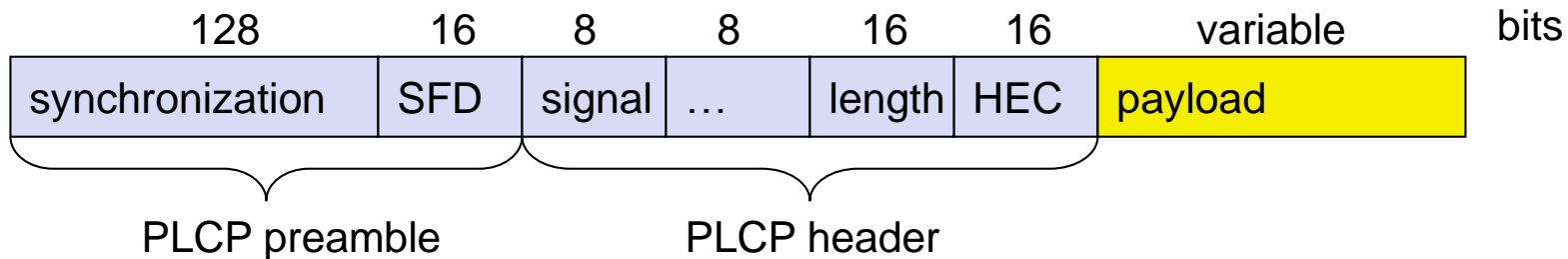
Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag	Address	Control	Protocol	Payload	Checksum	Flag
	01111110	11111111	00000011				01111110

**Byte stuffing**

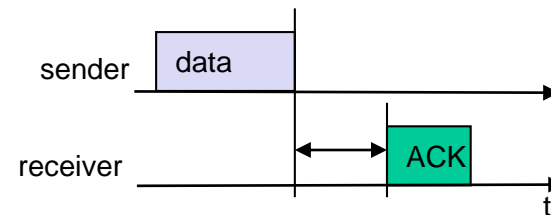
- ◆ Framing: Flags - 0x7E
- ◆ Byte stuffing: ESC – 0x7D
- ◆ Error detection – can be negotiated
- ◆ No ARQ



# Wireless LAN



- ♦ Framing
  - » Synchronization: 0101010 ...
  - » SFD (Start Frame Delimiter → 1111001110100000
  - » Length → Payload length **in us**
- ♦ HEC (Header Error Check)
  - » ITU-16,  $G(x) = x^{16} + x^{12} + x^5 + 1$
- ♦ Payload (data)
  - » Protected by strong codes
- ♦ ARQ
  - » modified version of Stop and Wait
- ♦ Signal: Payload bitrate (0A: 1 Mbit/s DBPSK; 14: 2 Mbit/s DQPSK)

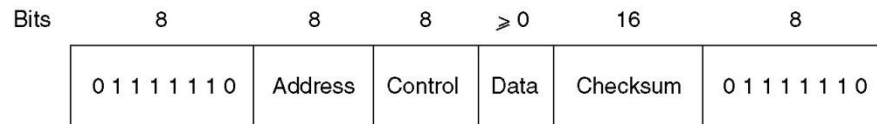




# *High-Level Data Link Control*

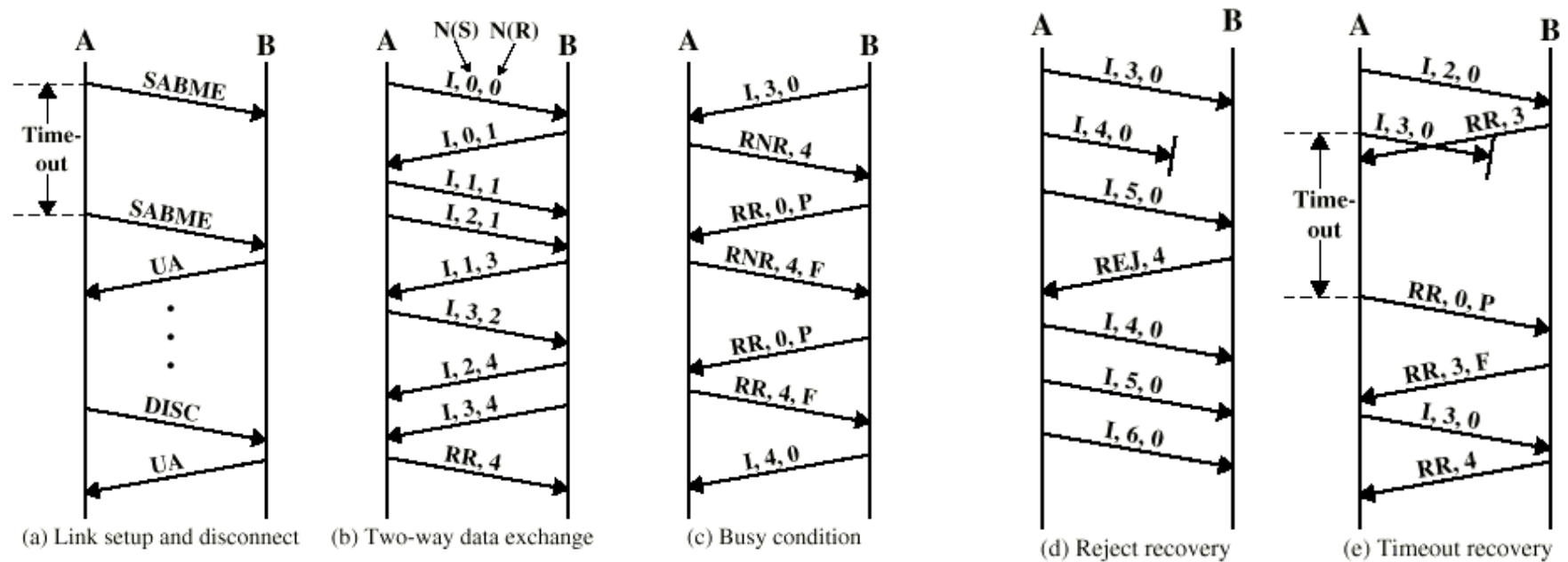
---

- ◆ HDLC, Data Link Control, bit oriented



- ◆ Framing – FLAGS
- ◆ Bit stuffing
- ◆ Error detection – ITU-16
- ◆ ARQ – Selective Repeat ARQ
- ◆ Used as basis for many telecom networks
  - » GSM/GPRS/UMTS, Frame Relay
  - » LAP-x protocols

# HDLC - Examples

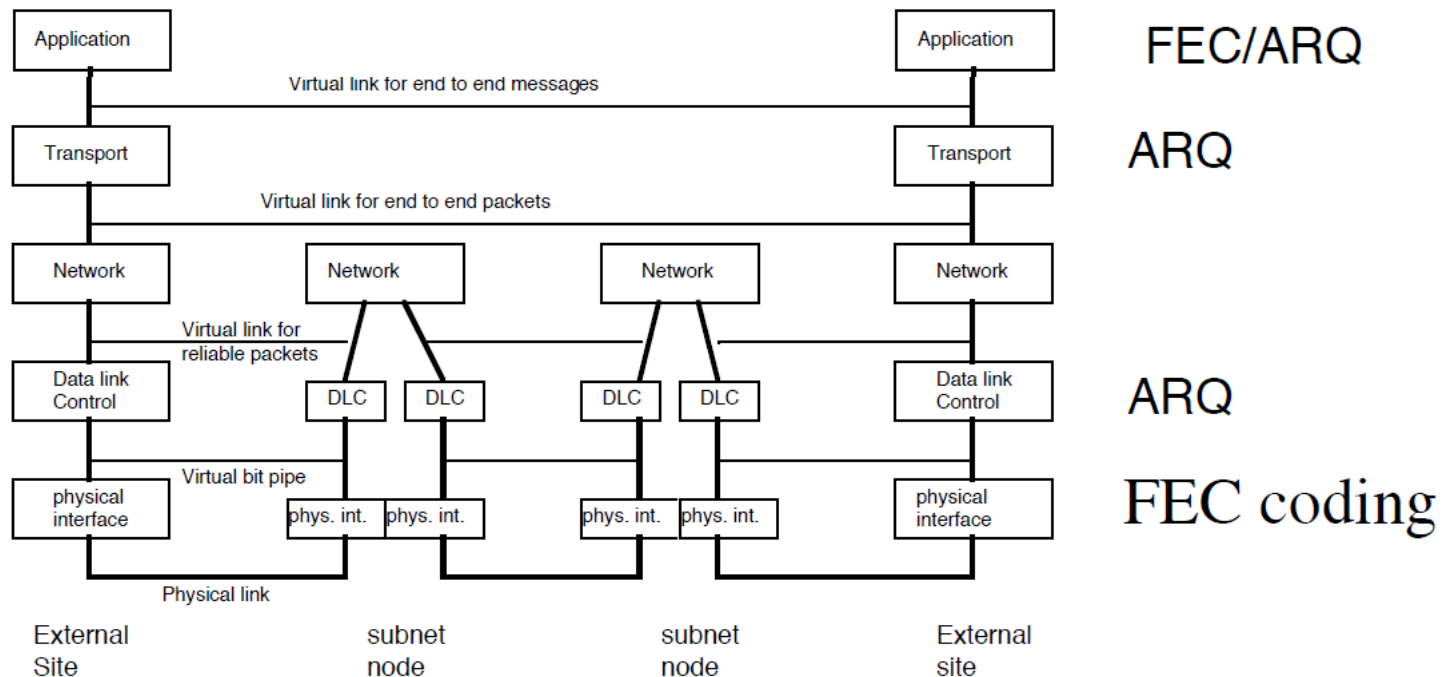


---

## *Reliability in the Protocol Stack*

# Reliability in the Protocol Stack


---



# *Reliability in the TCP/IP Reference Model*

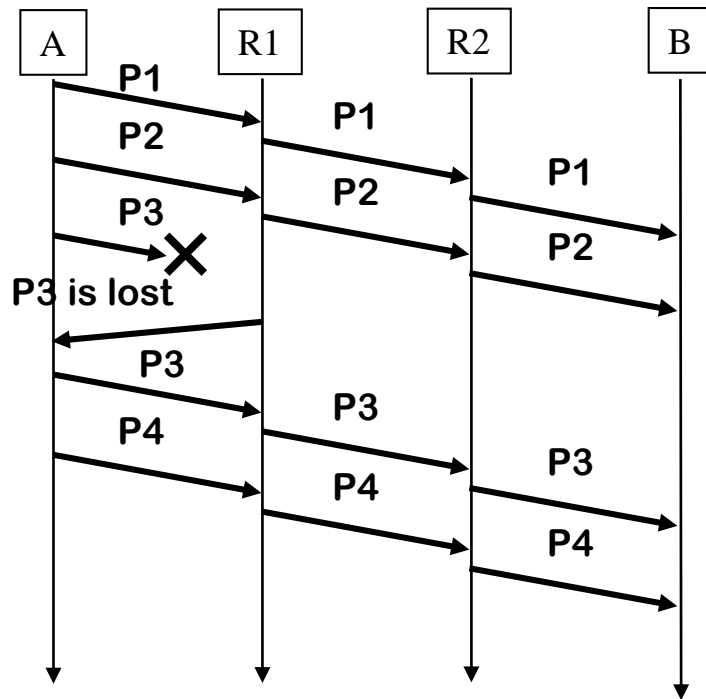
---

- ♦ The TCP/IP reference model assumes
  - » Every layer 2 offers an error free service to the upper layer
  - » Service Data Units are
    - delivered to upper layer without error,
    - or discarded
- ♦ The layered model **transforms bit error in packet losses**  
Therefore, packet losses must be repaired ➔ ARQ solutions
- ♦ Two strategies can be used
  - » Link-by-Link ARQ
  - » End-to-end ARQ

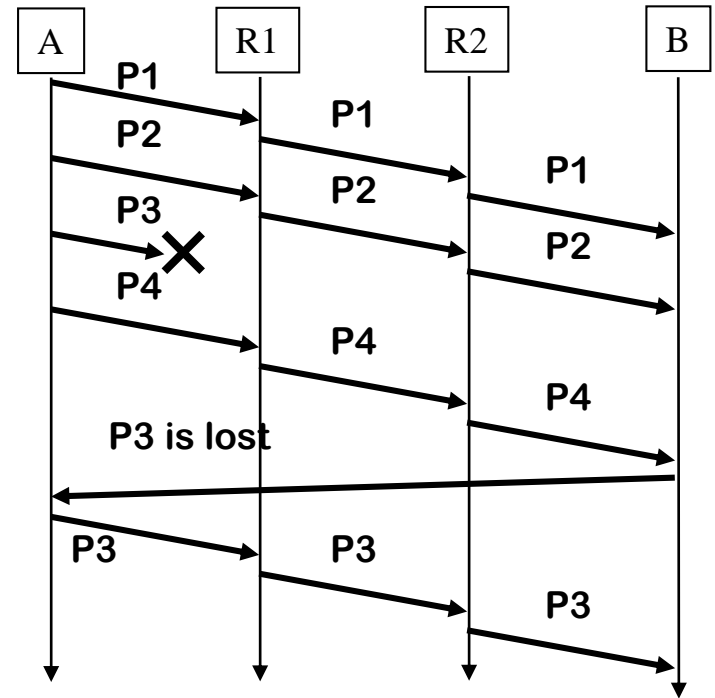

$$FER = 1 - (1 - BER)^n$$

# *Link-by-Link ARQ versus End-to-End ARQ*

---



Link-by-Link ARQ  
(data link layer)



End-to-End ARQ  
(transport or Application layers)

# *Link-by-Link ARQ versus End-to-End ARQ*

---

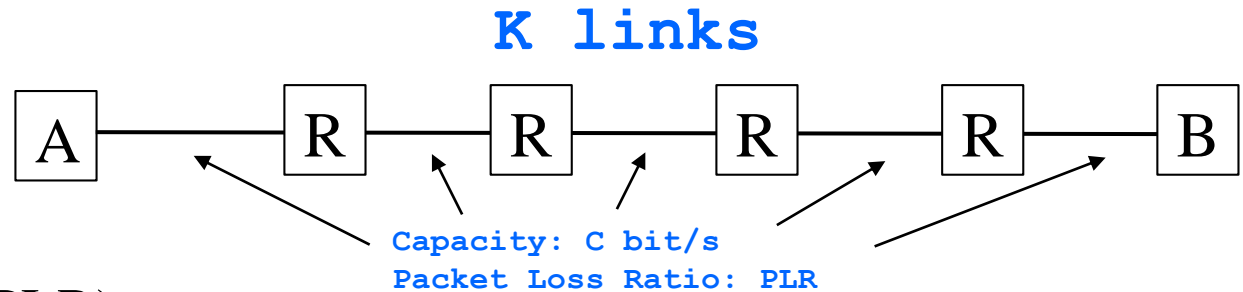
## ♦ Link-by-Link ARQ

- » Repairs losses link by link
- » Requires network elements to
  - remember information about packet flows → high processing per frame/packet
  - store packets in case they have to be retransmitted → memory required

## ♦ End-to-end ARQ

- » Low complexity in intermediate network elements
  - Switches/routers become simpler
- » Packets may follow different end-to-end paths
- » But, not acceptable when Packet Loss Ratio is high
- » Let's see why ...

# End to End Capacity



- ◆ Packet Loss Ratio (PLR)
  - » Packet → layer 3; Frame → layer 2
  - » Let's assume PLR=FER  
(not considering losses in queues)
- ◆ Capacity of one link  $C_l = C * (1 - PLR)$
- ◆ End to End capacity
  - » using Link-by-Link ARQ:  $C_{LL} = C_l = C * (1 - PLR)$
  - » Using End-to-End ARQ:  $C_{EE} = C * (1 - PLR)^K$
- ◆ End-to-end ARQ → **Inefficient when PLR is High**

k	PLR	$C_{EE}$	$C_{LL}$
10	0.05	<b><math>0.6 * C</math></b>	$0.95 * C$
10	0.0001	$0.9990 * C$	$0.9999 * C$



# *ARQ in the TCP/IP Reference Model*

---

- ◆ The TCP/IP architecture assumes that
  - » The Data Link layer **provides error-free packets** to the network layer
  - » Data link layer provides a service with very low FER
  - » End-to-End ARQ is used, implemented at  
Transport or Application layers
- ◆ In the TCP/IP reference model, packet losses are repaired
  - » At the Data Link layer on lossy channels (e.g. wireless data links)
  - » At the end systems (transport layer or application layer)

# *Homework*

---

1. Review slides
2. Read from Tanenbaum
  - » Chapter 3 (5<sup>th</sup> edition)
3. Read from Bertsekas&Gallager
  - » Sections 2.3, 2.4
4. Answer questions at moodle