

ETSF15: Lecture 3

- L2 Error and Flow Control
 - Framing
- Performance

Jens A Andersson



This is our quest!

Make an application communicate over a distance



Hi!

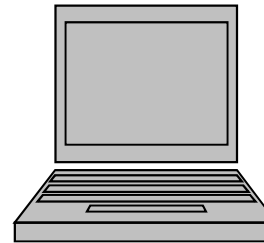
Hi!

Buy milk, please.

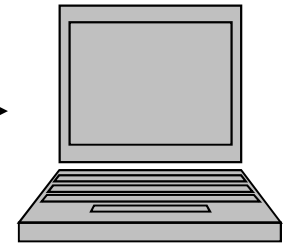
Will do!

Bye!

Bye!



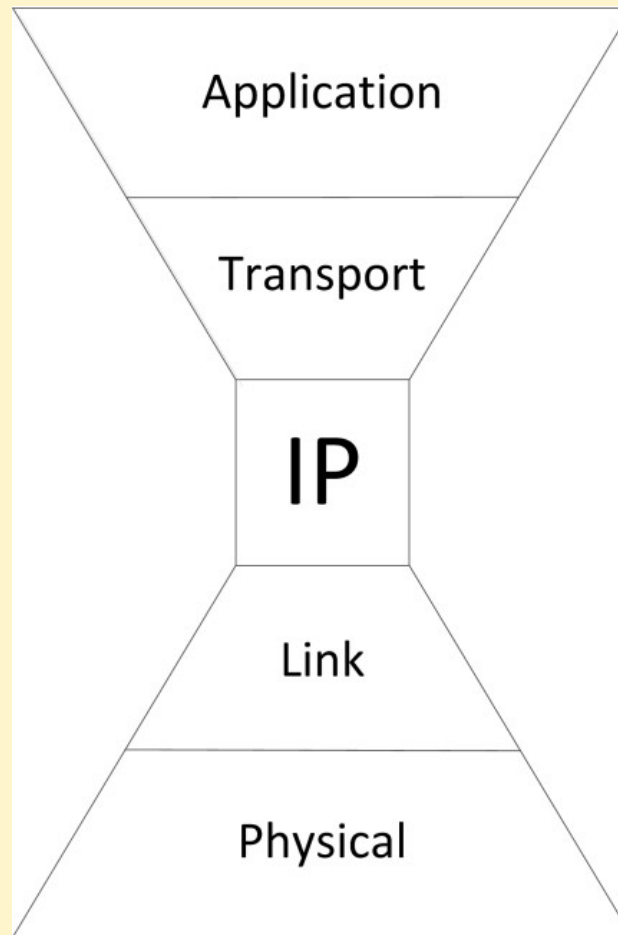
11001000101



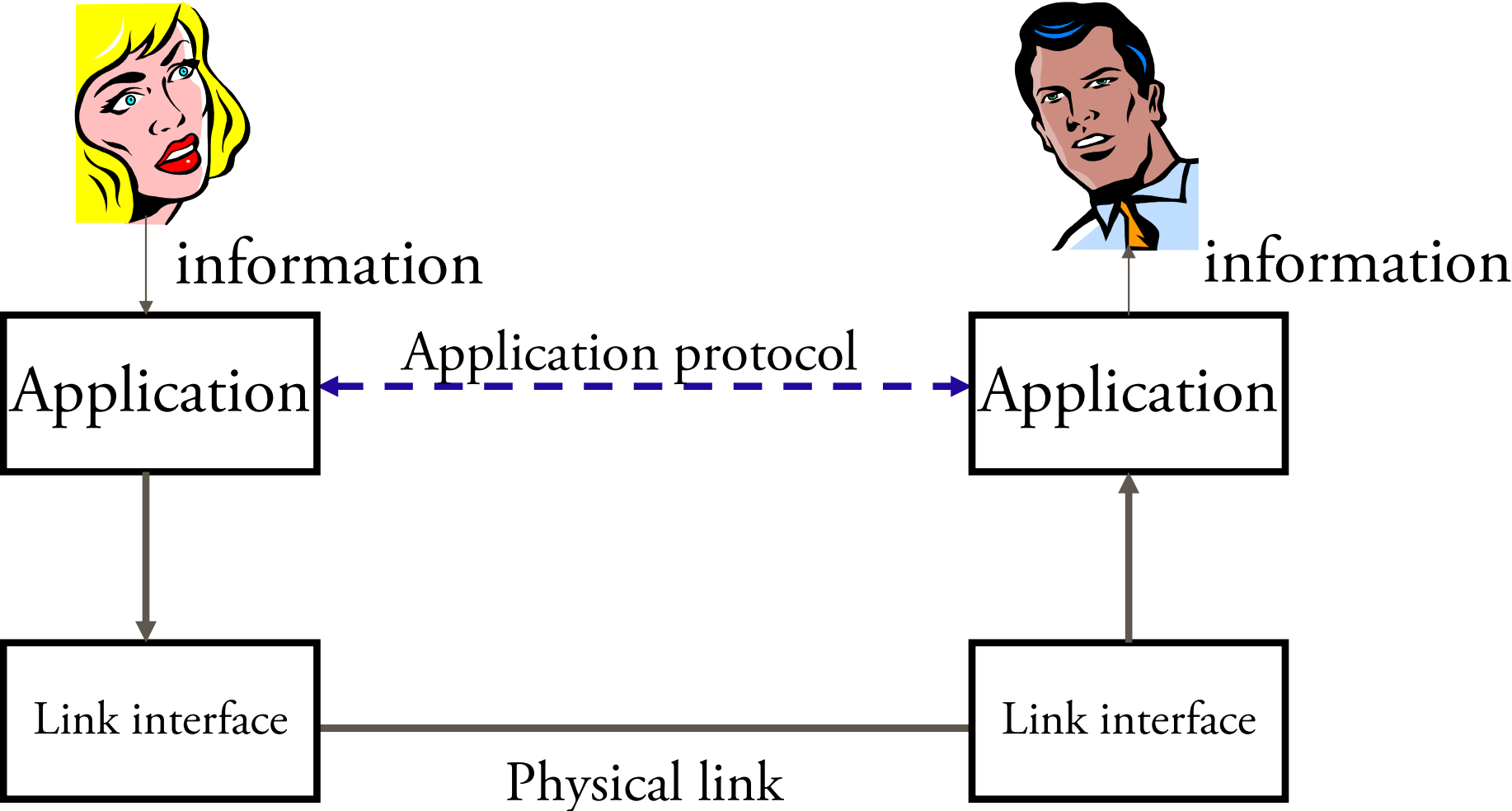
Layered network models

- Too complicated to solve everything in one application
 - ◆ *Divide and conquer*
 - ◆ Hierarchical
 - ◆ Specialising
 - ◆ Simplifying

Actual reference model: Hour glass model



Protocols



HTTP, an application protocol

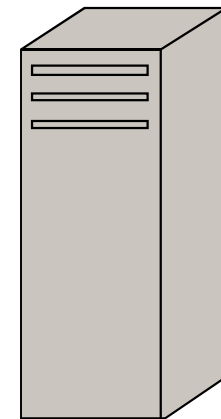
- Hyper Text Transfer Protocol = HTTP



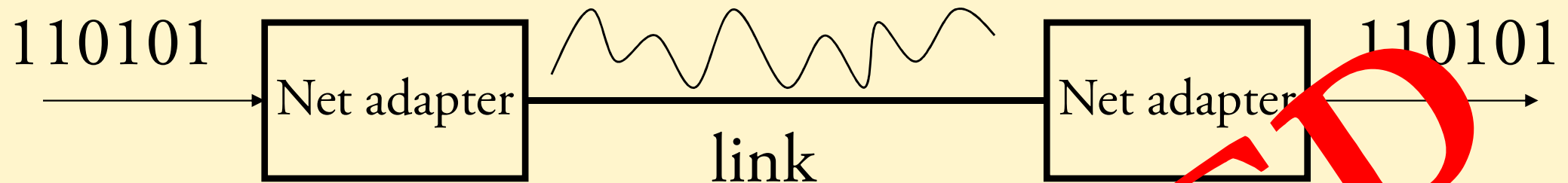
HTTP request



HTTP reply



Problem 1: Digital communication



- In the transmitter there is an adapter that converts bits into signals that are then sent on the link.
- An adapter in the receiver translates the signals into bits again.

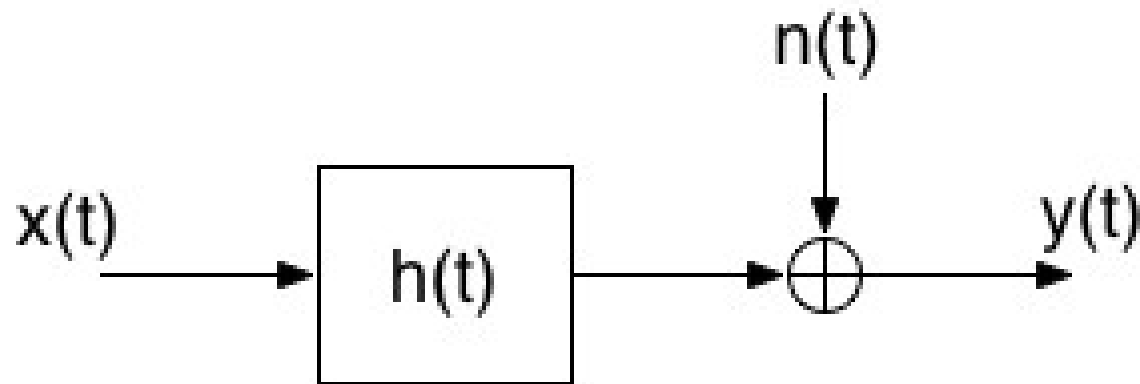
SOLVED

Problem 4

- How to detect transmission errors (and correct)?
 - ◆ Stupid to transfer 1GB only to discover that a bit error occurred

Error control

Find errors in transmitted data?



$$y(t) = x(t) \text{ for all } t?$$

$x(t) : \dots 0110010100100011110 \dots$

Use more than one channel,

compare received $y_1, y_2, y_3 \dots$

L2: Error control

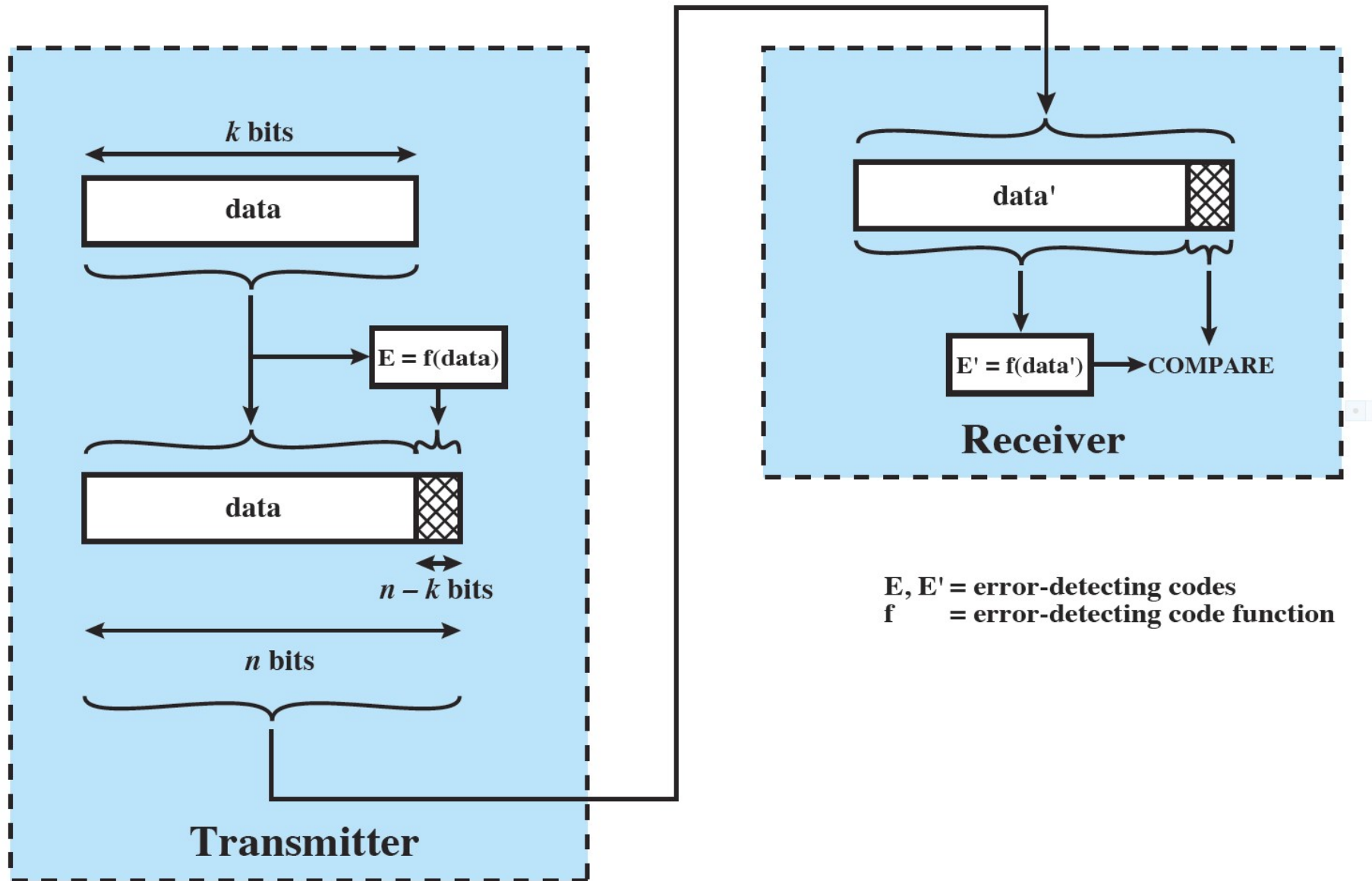
Solution: Frames/Packets!

$$x(t) : 0110010100100011110$$

(t is a limited number)

- Finite number of bits per frame
- Add extra bits to each frame:
 - ◆ Parity bits
 - ◆ CRC: Cyclic Redundancy Check

Error detection process



Parity bit

- Sender adds one bit to vector
 - ◆ Even parity: even number of 1s in new vector
 - ◆ Odd parity: odd number of 1s in new vector

$$\boxed{10011100} + \boxed{0} = \boxed{100111000}$$

Modula 2 Arithmetic

\oplus	0	1
0	0	1
1	1	0

\otimes	0	1
0	0	0
1	0	1

$$1 \oplus 1 = 0$$

$$1 - 1 = 0$$

Polynom represents vector

$$\mathbf{a} = a_{L-1}a_{L-2} \dots a_1a_0$$

$$\begin{aligned} a(x) &= \sum_i^{L-1} a_i x^i \\ &= a_{L-1}x^{L-1} + a_{L-2}x^{L-2} + \dots + a_0x^0 \end{aligned}$$

$$\text{Number of bits} = \text{deg}(a) + 1$$

Adding 'parity' bits

Data to be transmitted:

$$d(x) = d_{k-1}x^{k-1} + d_{k-2}x^{k-2} + \cdots + d_1x^1 + d_0$$
$$\deg(d) = k - 1$$

Add $n - k$ bits giving a codeword of length n :

$$r(x) = r_{n-k-1}x^{n-k-1} + \cdots + r_1x^1 + r_0$$
$$\deg(r) = n - k - 1$$

Codeword:

$$c(x) = d(x)x^{n-k} + r(x)$$

Find $r(x)$

Use generator polynomial:

$$g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_1x + 1$$

Note:

$$\deg(g) = n - k = \deg(r) + 1$$

$$g_{n-k} = 1$$

$$g_0 = 1$$

$$\mathbf{r(x) = R_{g(x)}(d(x)x^{n-k})}$$

Theorem

A polynomial $c(x)$ with $\deg(c(x)) < n$ is a codeword if and only if $g(x) | c(x)$.

$g(x) | c(x) = c(x)$ is a multiple of $g(x)$

At the receiver side

Received codeword:

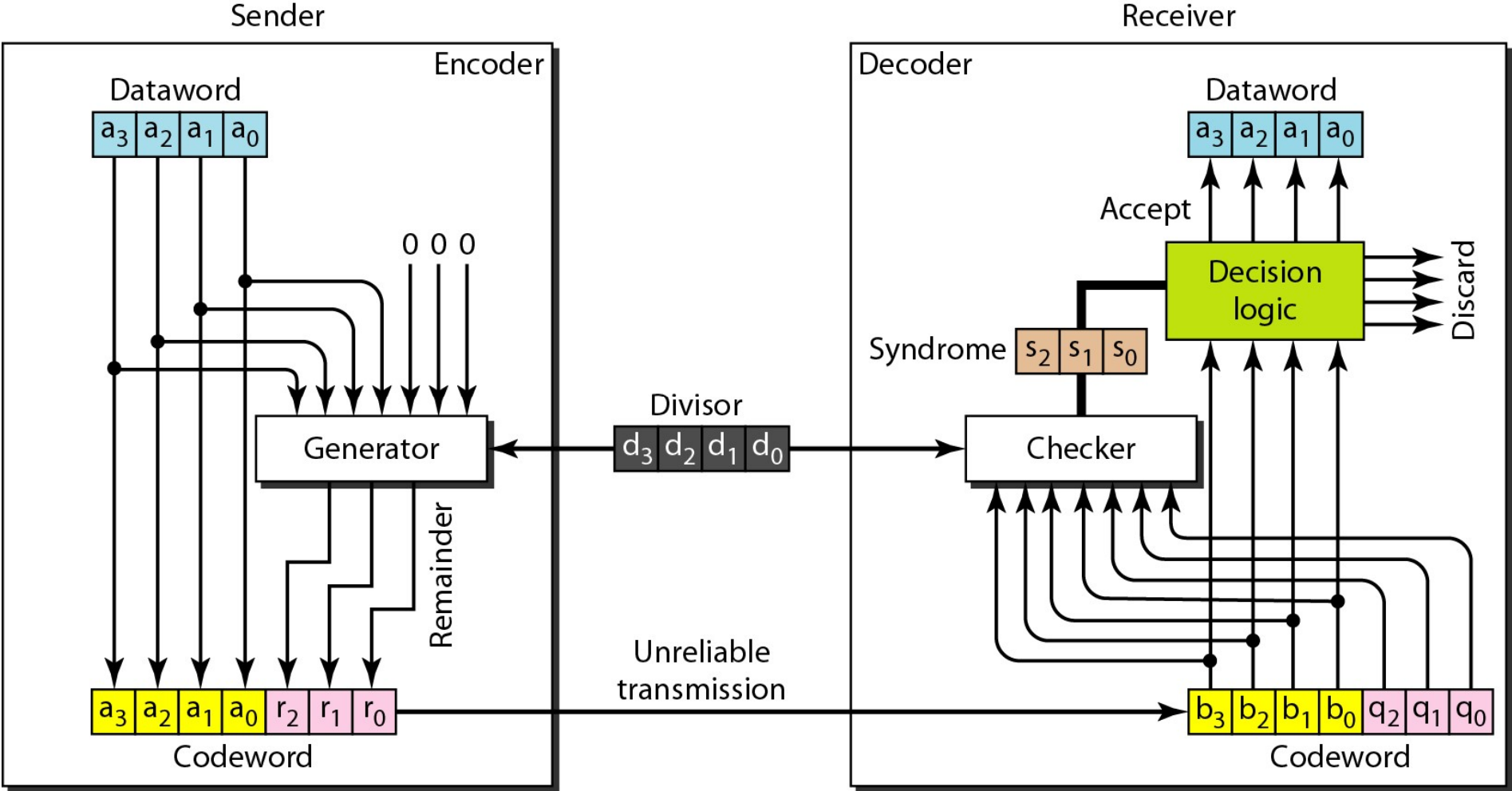
$$y(x) = c(x) + e(x)$$

Calculate *syndrome* of received vector:

$$\begin{aligned} s(x) &= R_{g(x)}(y(x)) = R_{g(x)}(c(x) + e(x)) \\ &= R_{g(x)}\left(R_{g(x)}(c(x)) + R_{g(x)}(e(x))\right) \\ &= R_{g(x)}(e(x)) \end{aligned}$$

- $R_{g(x)}(c(x))=0$ (see Theorem)
- $s(x) = 0$ transmission OK!

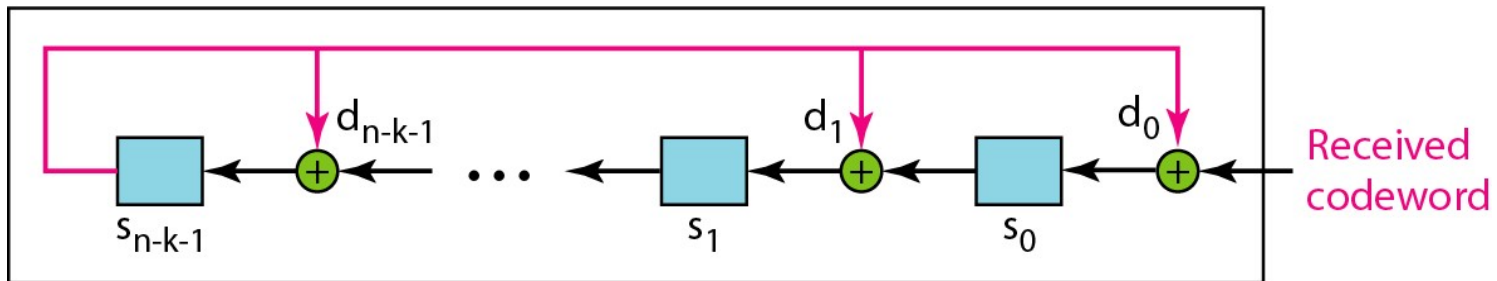
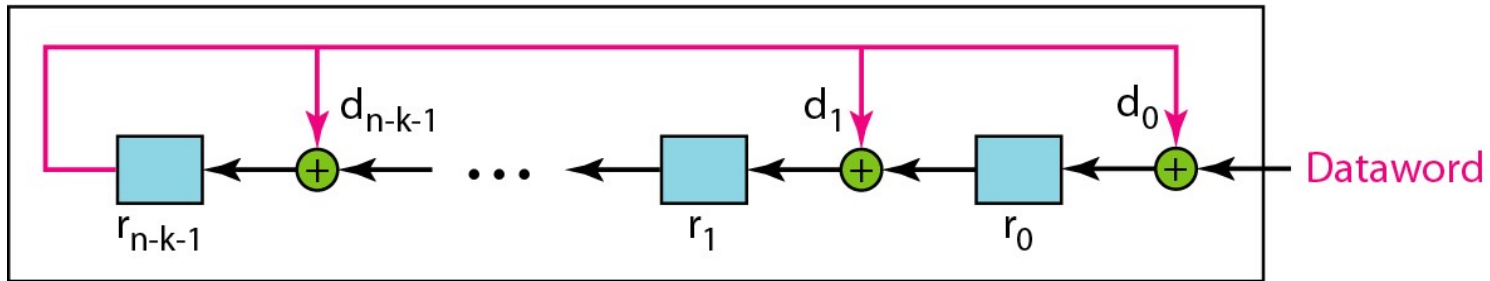
CRC block diagram



CRC division in the digital domain

Note:

The divisor line and XOR are missing if the corresponding bit in the divisor is 0.



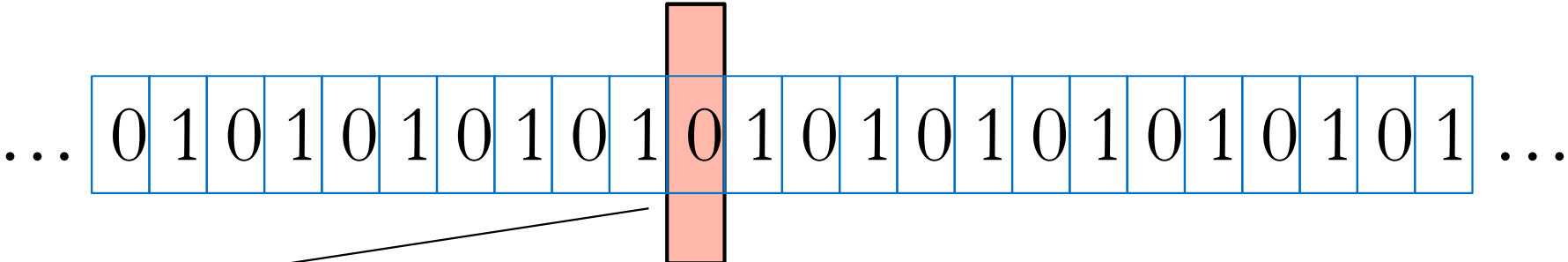
L2: Error correction

- Forward Error Correction:
 - ◆ Add extra bits so a limited number of errors can be fixed
 - ◆ Costly
 - ◆ What to do with errors that can be detected but not fixed?
- Retransmit
 - ◆ Automatic Repeat reQuest ARQ

Automatic Repeate reQuest (ARQ)

- All sent frame has to be acknowledged (ACK) before sending next frame(s)
- Three versions:
 - ◆ Stop-And-Wait
 - ◆ Go-Back-N
 - ◆ Selective-Repeate
- Use Sliding Window
 - ◆ Sender keeps track of sent and ACKed frames
 - ◆ Receiver keeps track of received frames

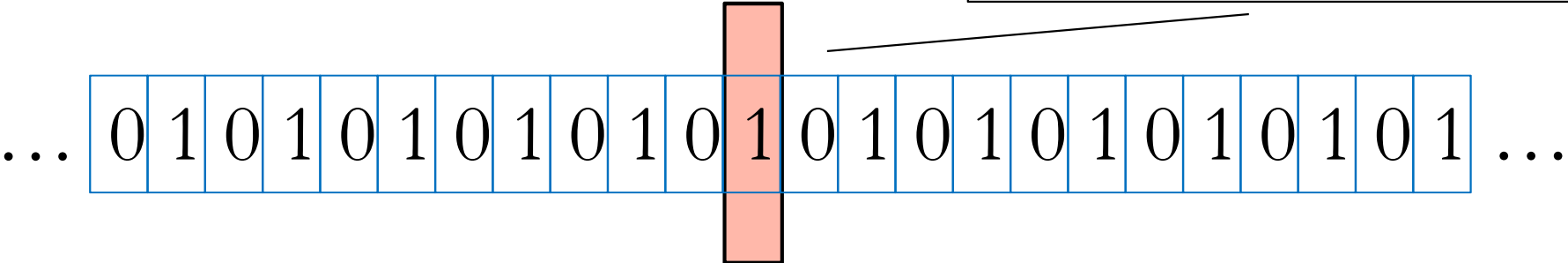
Sender Sliding Window



Before sliding

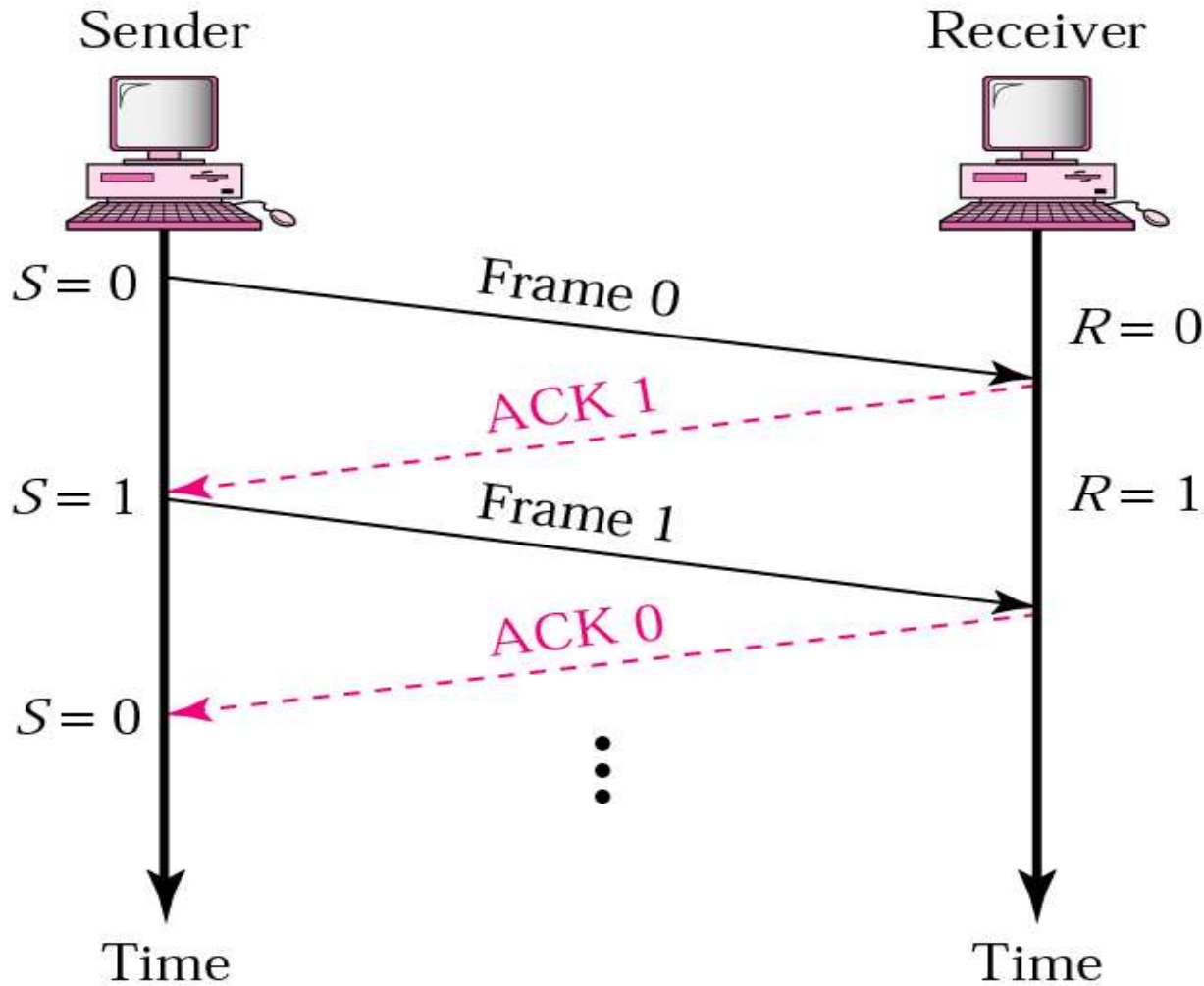
ACKs move
this pointer

Sent frames move this pointer if
actual win size \leq max win size



After sliding on frame

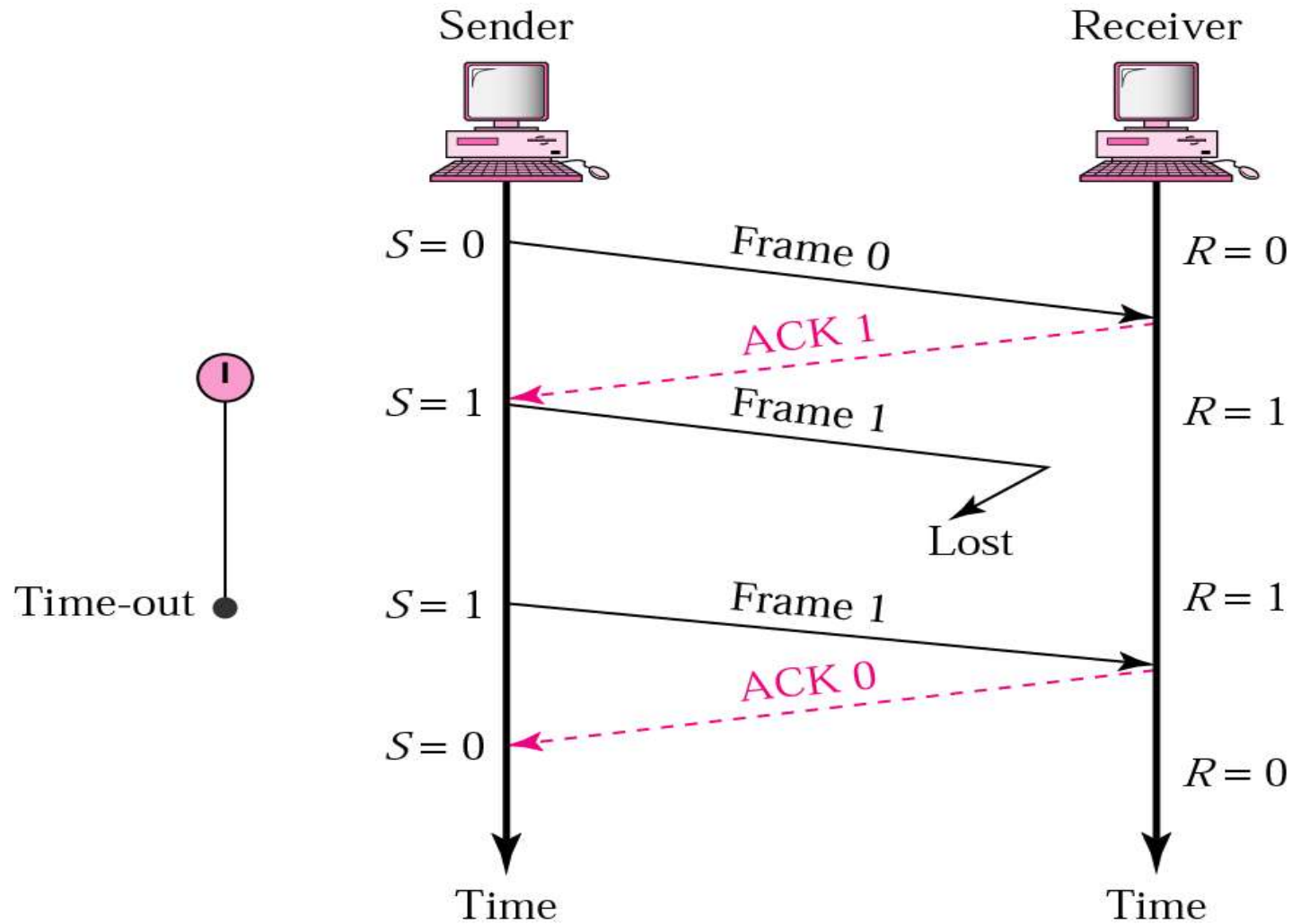
Stop-and-Wait Normal operation



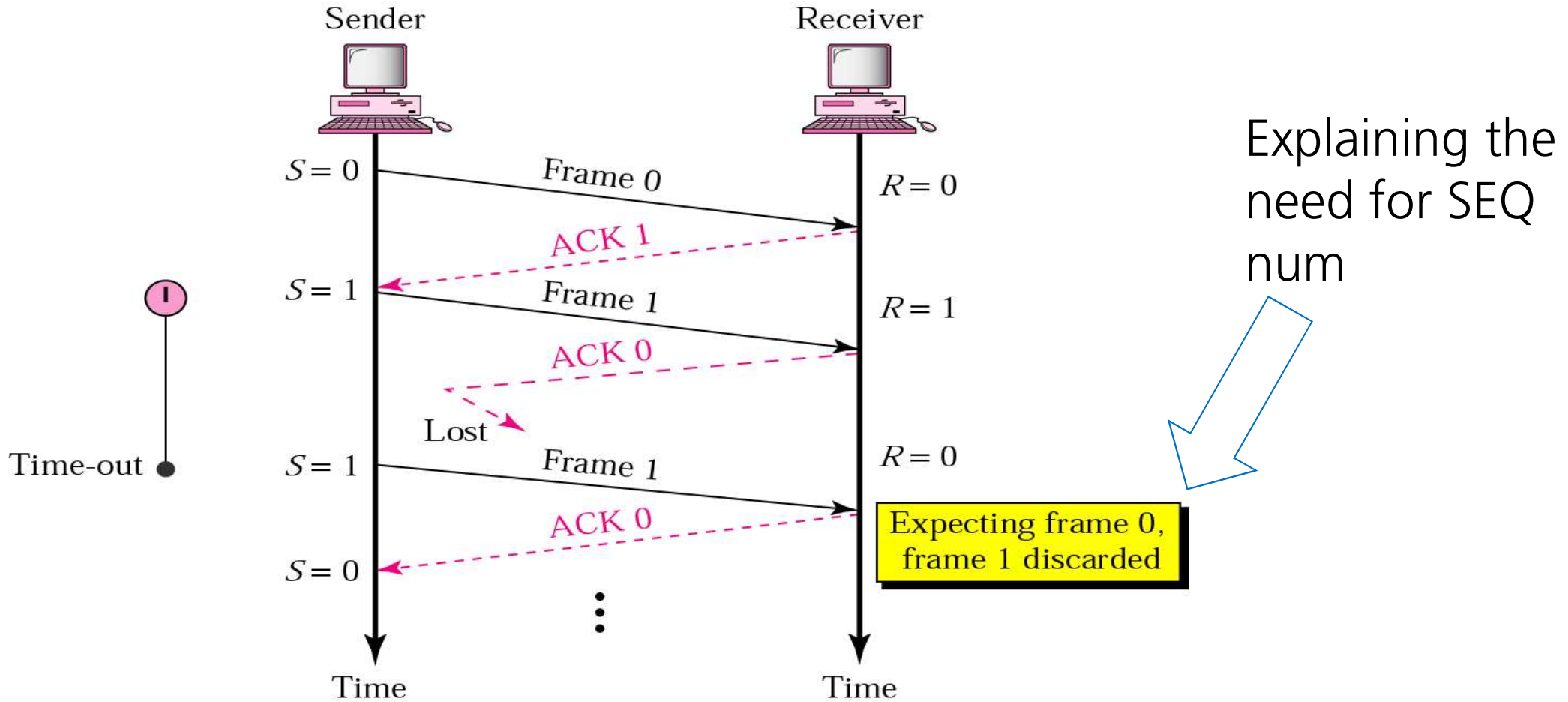
Note!

- Sequence numbers
- Sliding Window size = 1 frame

Stop-and-Wait ARQ, frame lost



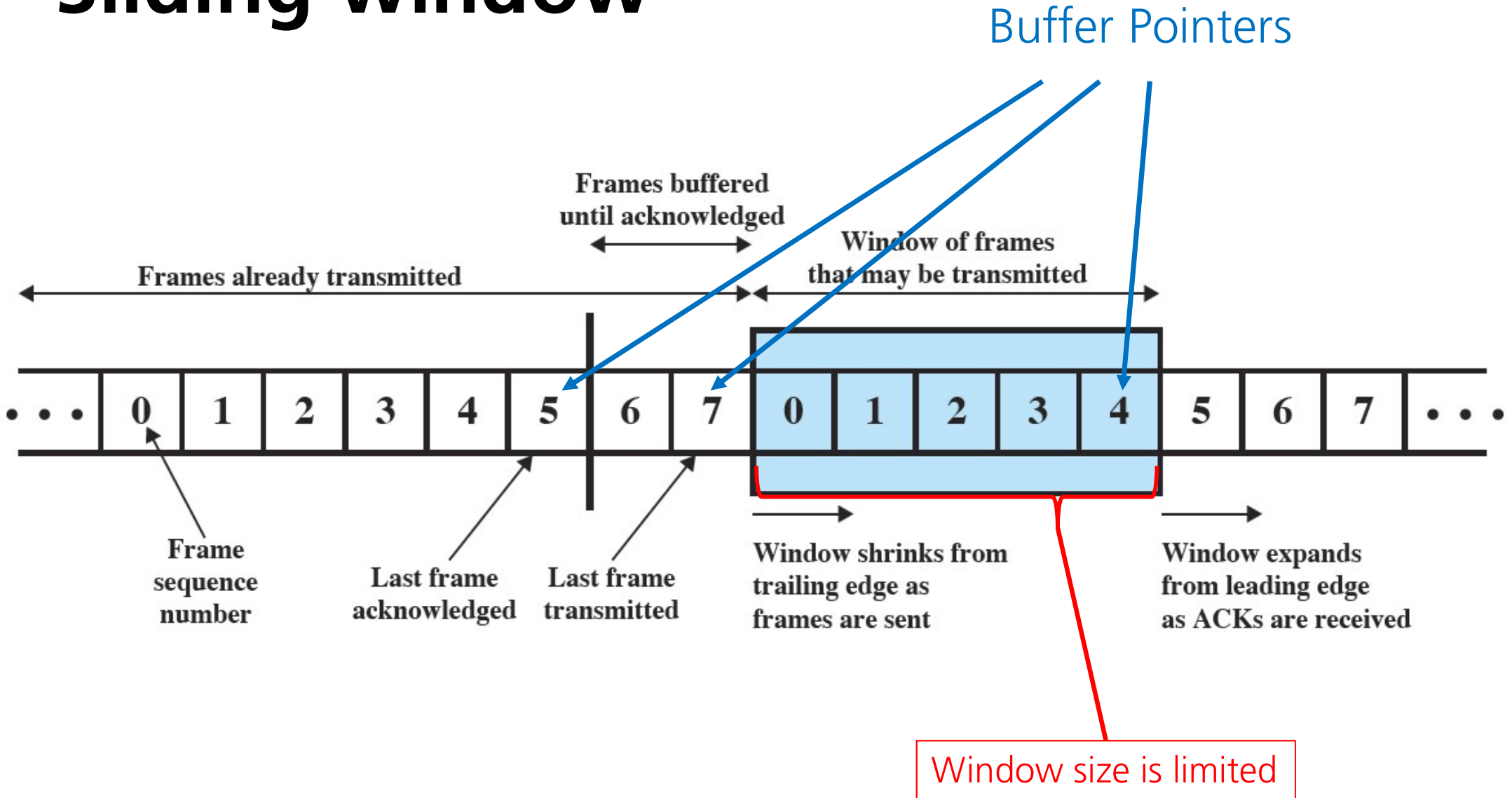
Stop-and-Wait ARQ, lost ACK frame



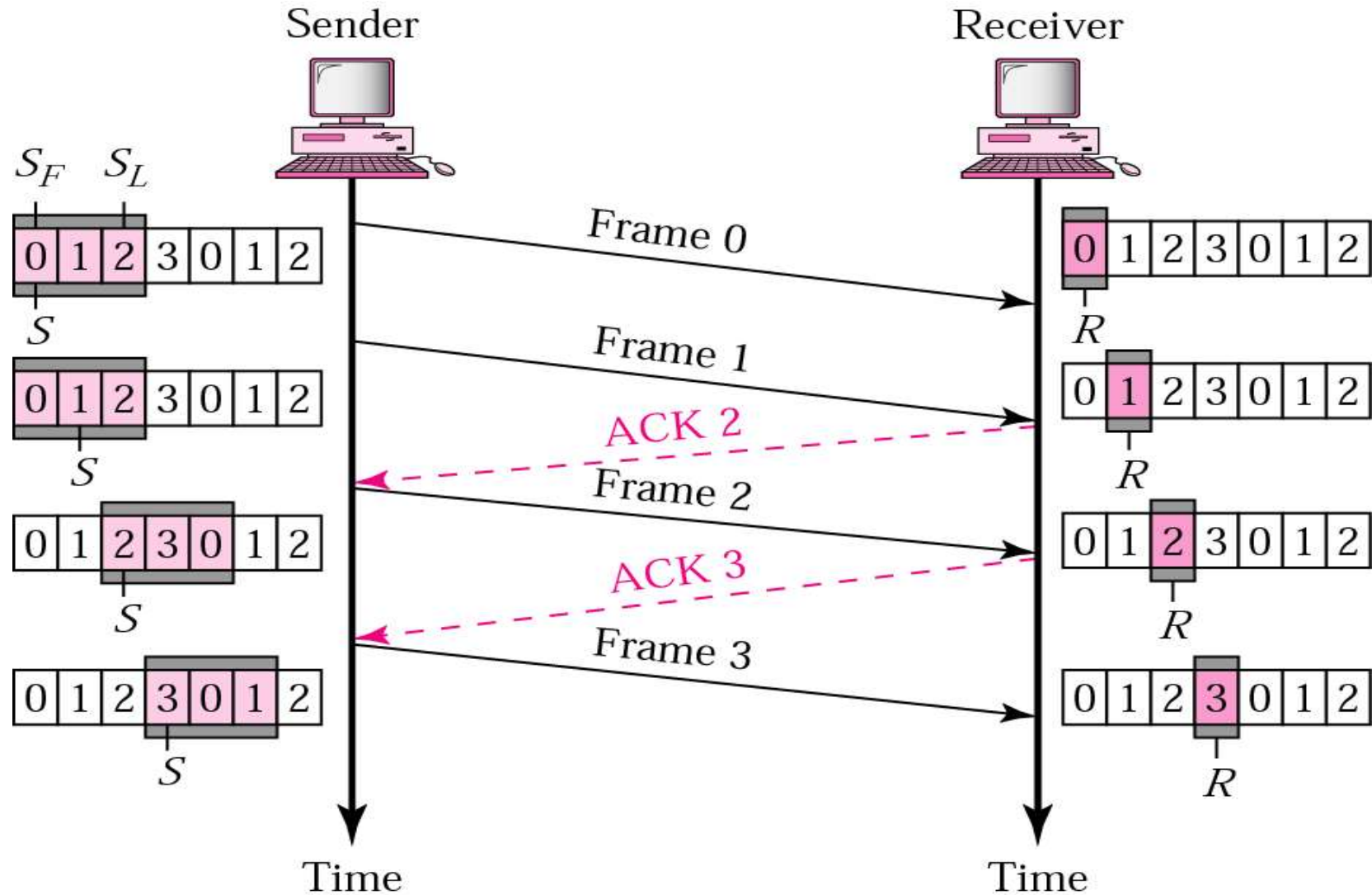
Go-Back-N

- Increase sliding window size
 - ◆ Sender can send as long as the sliding window includes frames not sent
 - ◆ Retransmitt requested frames and all following frames

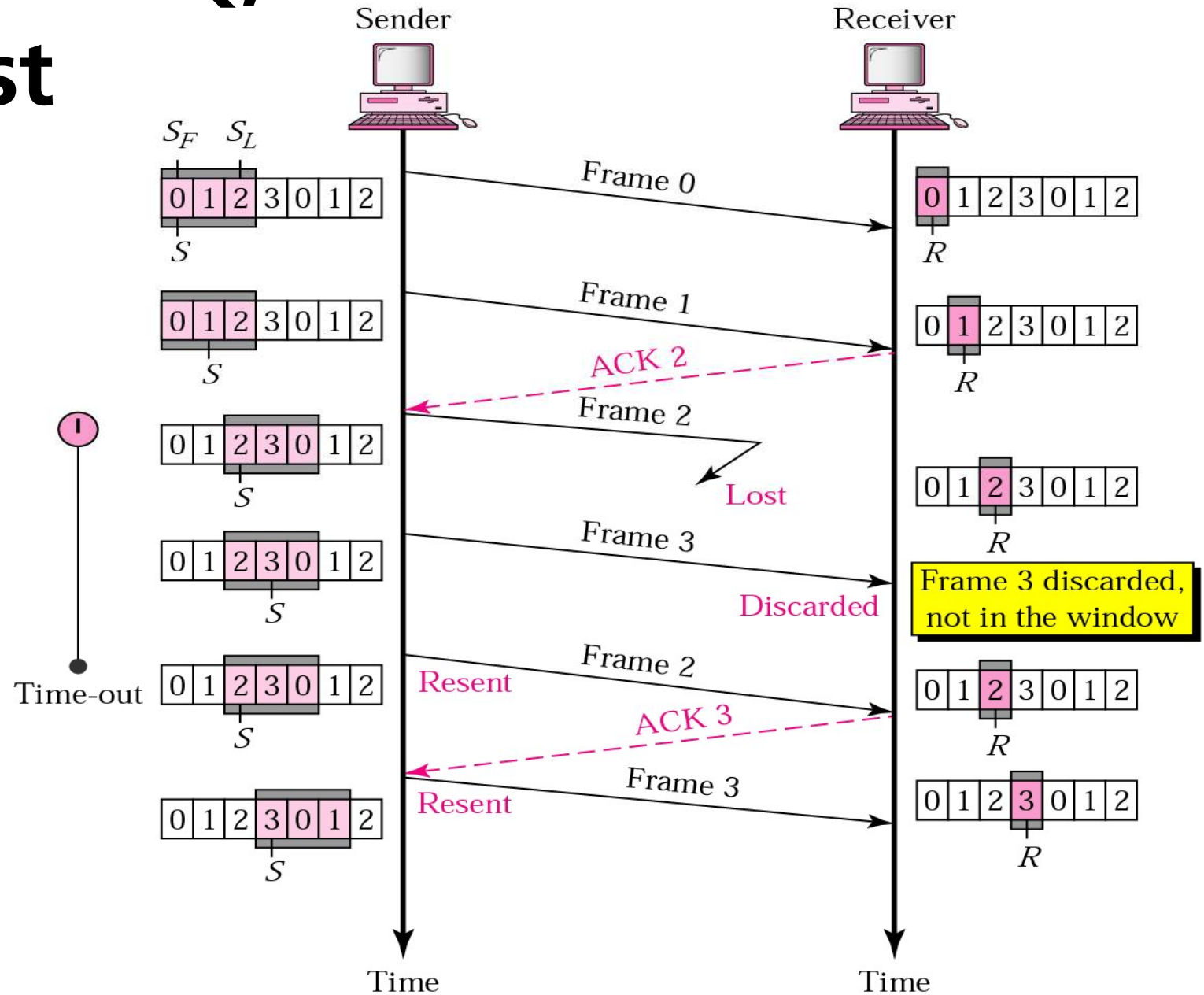
Sliding window



Go-Back-N ARQ, normal operation



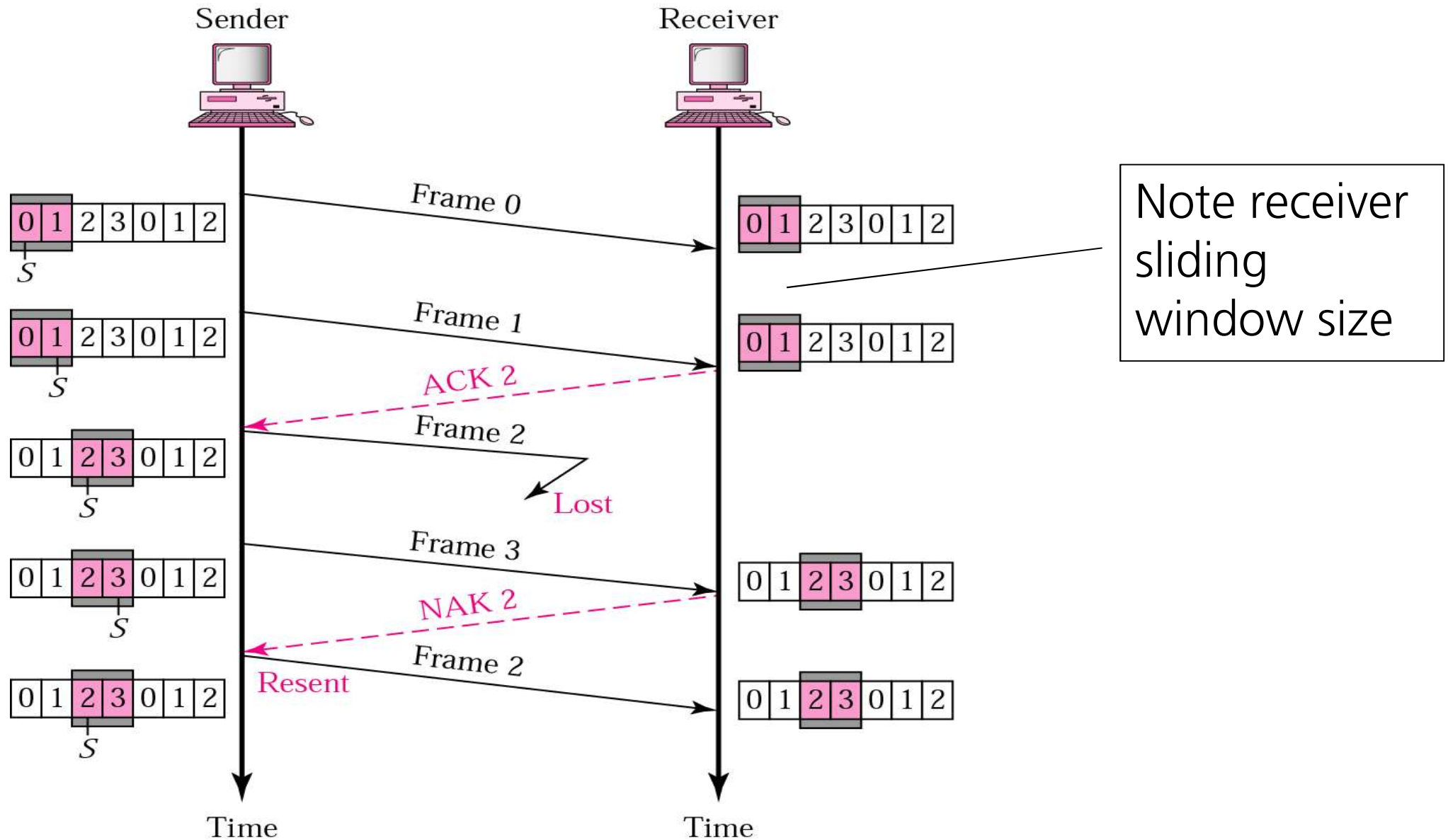
Go-Back-N ARQ, frames lost



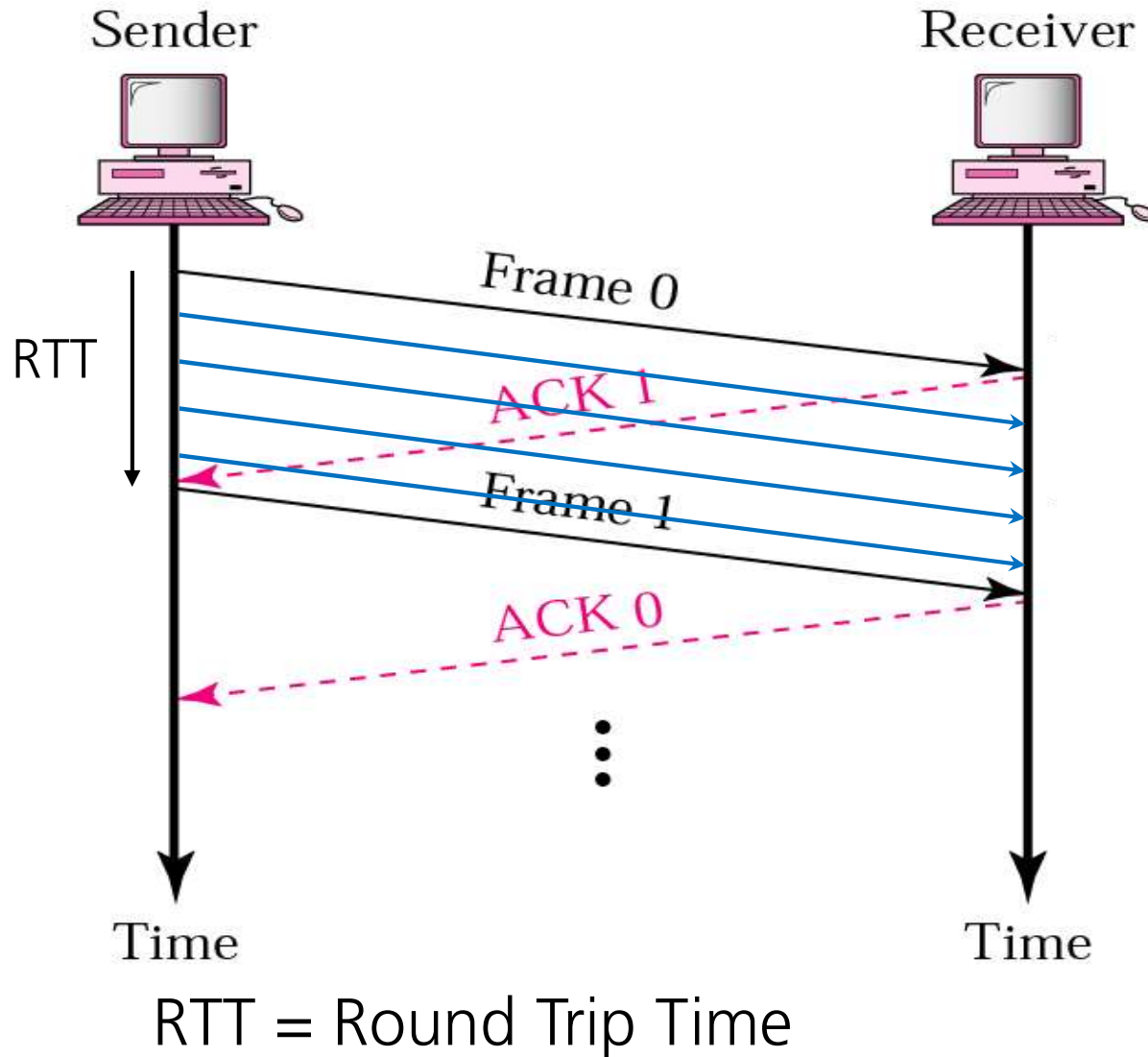
Selective-Repeat

- Same as Go-Back-N but
- Retransmitt only requested frames
- More efficient regarding network utilisation
- Higher demands on receiver and sender
 - ◆ Receiver must have bigger buffer

Selective Repeat ARQ, lost frame



L2: Flow control



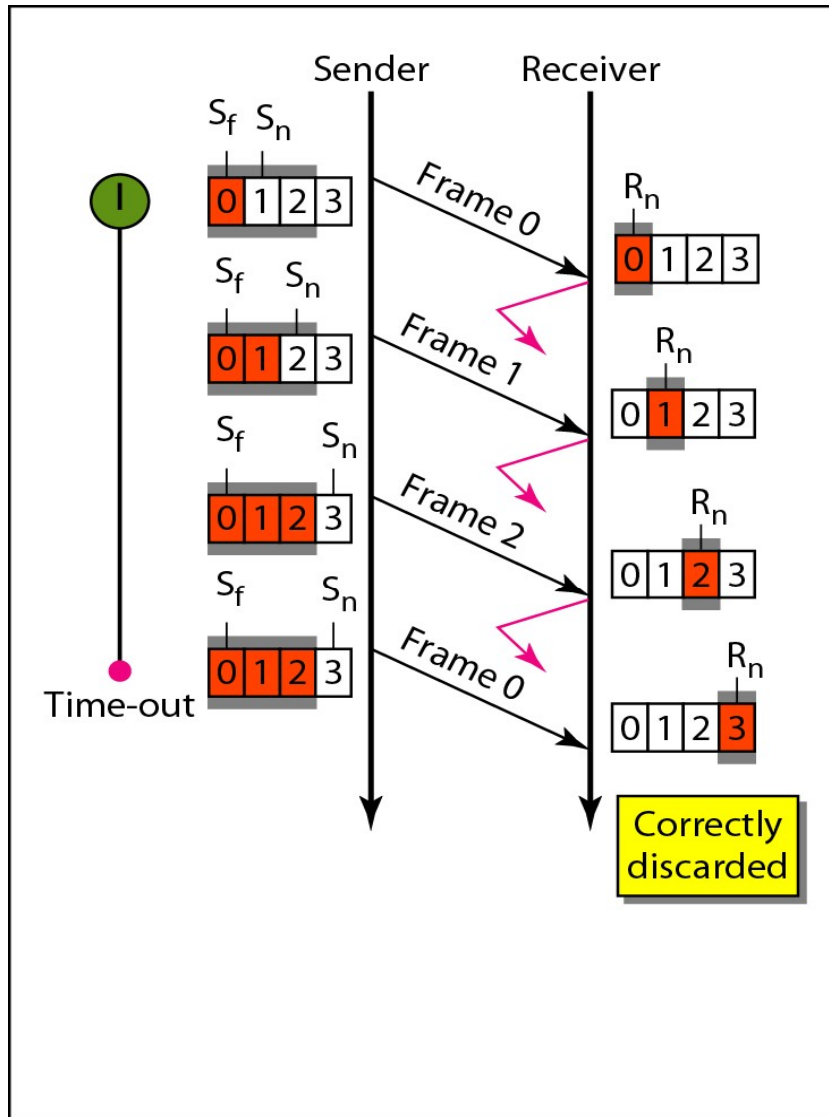
Idea:

- Assume error free transmission
- Allow frames to be sent until ACK for first frame is expected = RTT
- Check RTT during transmission
- Thus, sliding window size = $f(\text{RTT})$

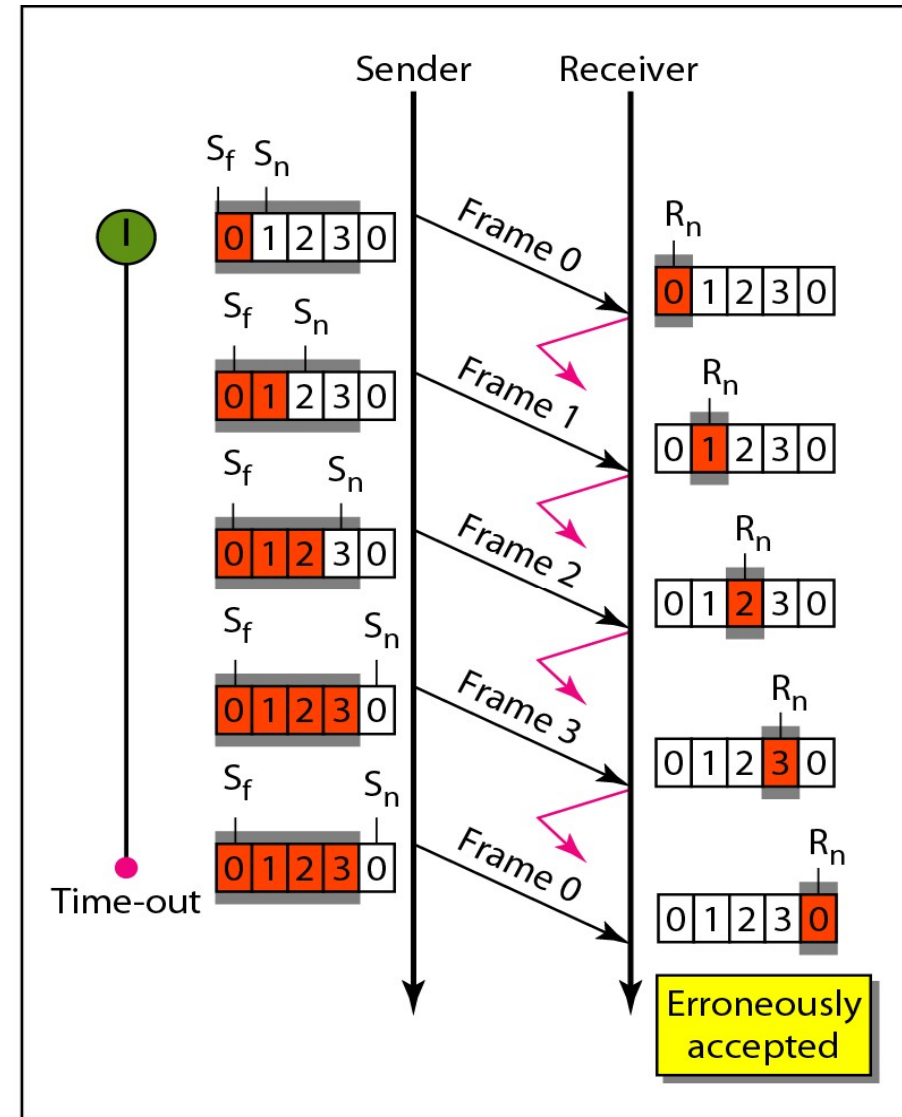
Some notes

- Piggy backing
 - ◆ Data and ACKs can share frame
- The number of bits for the sequence number is a function of the max window size
 - ◆ Seq numbers wrap!

Window size for Go-Back-N ARQ

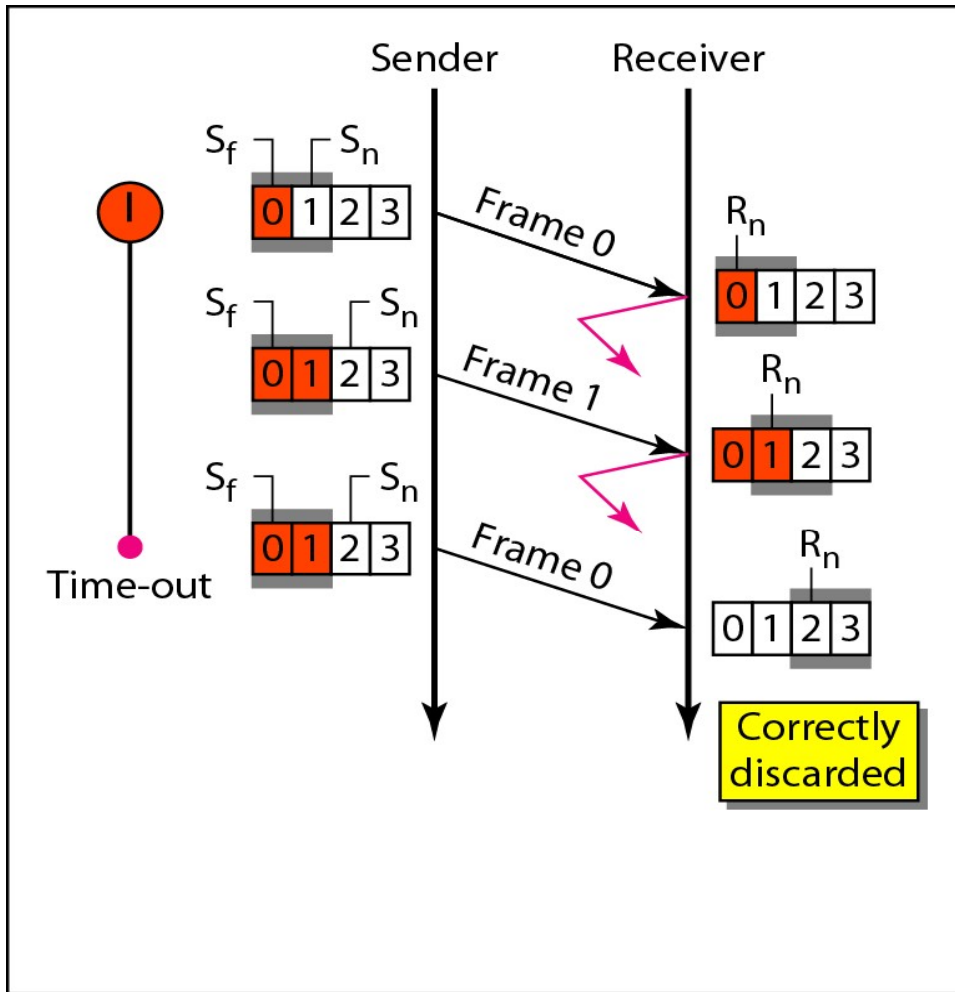


a. Window size $< 2^m$

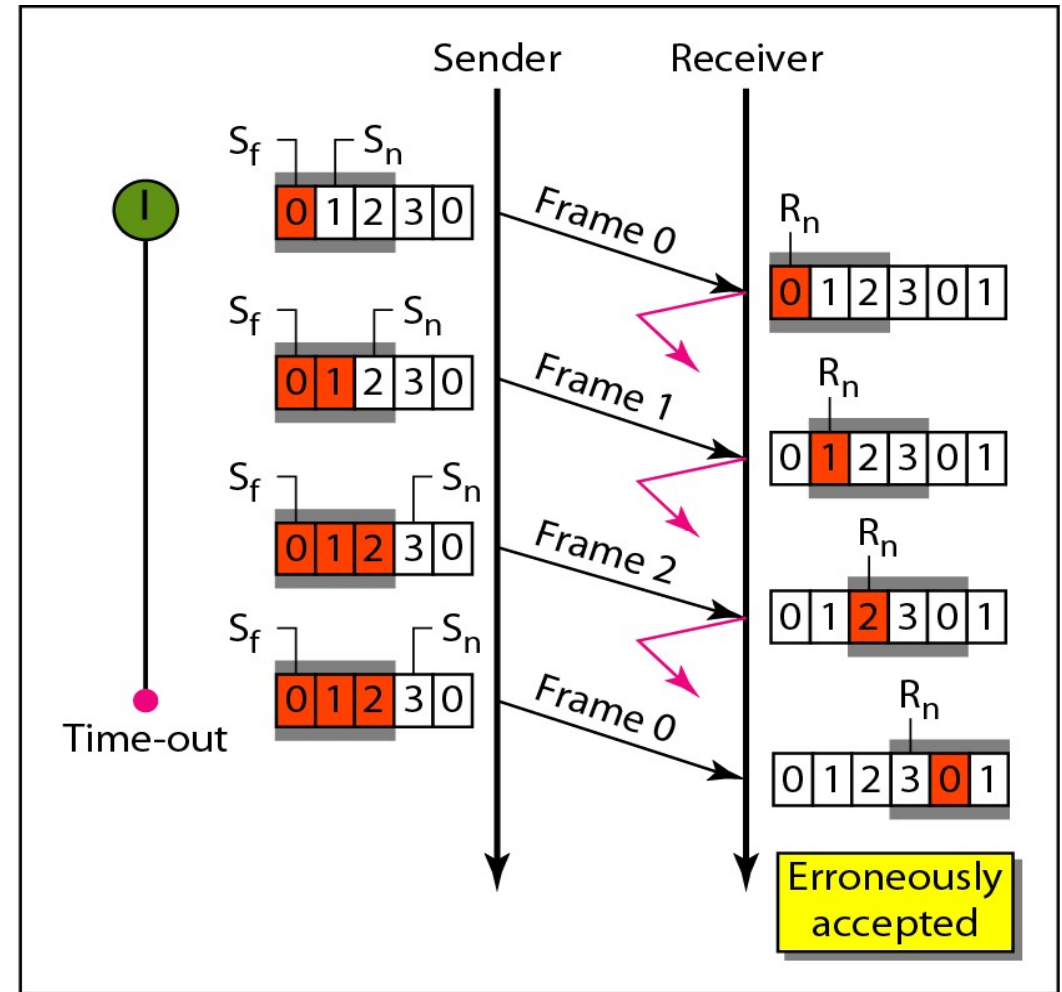


b. Window size $= 2^m$

Selective Repeat ARQ, window size



a. Window size = 2^{m-1}



b. Window size $> 2^{m-1}$

Framing



- Header:
 - ◆ Sequence and ACK numbers
 - ◆ More to come ...
- Tail
 - ◆ CRC

Synchronisation

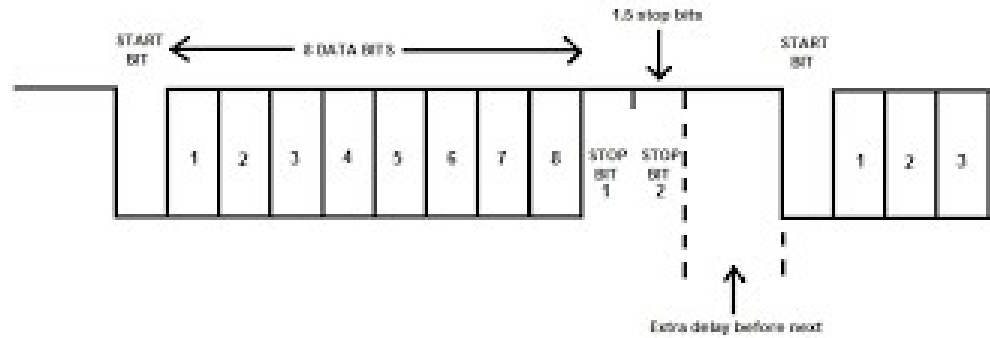
Preamble and start flag

Frame

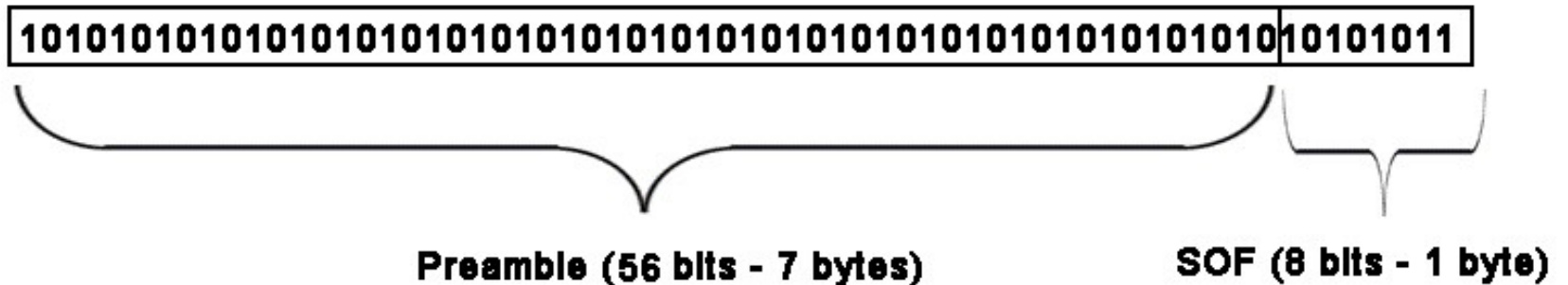
End flag

Receiver has to synchronise to signal of a frame

RS.232/V.24



Ethernet




Finding the start flag

- Corrolate incoming bit pattern with know flag
- If deploying end flag we have a problem
 - ◆ What if the end flag bit patterns = data bit pattern?

Bit stuffing

- Given: Flag = 01111110
- Task: Avoid 6 consecutive bits = 1 in payload
- Solution:
 - ◆ Sender: In payload add a 0 after 5 consecutive bits = 1
 - ◆ Receiver: Remove bit following 5 consecutive bits = 1

011111101111100111000111111

011111**0**1011111**0**0011100011111**0**1

One link layer protocol: HDLC

- HDLC = High-level Data Link Control



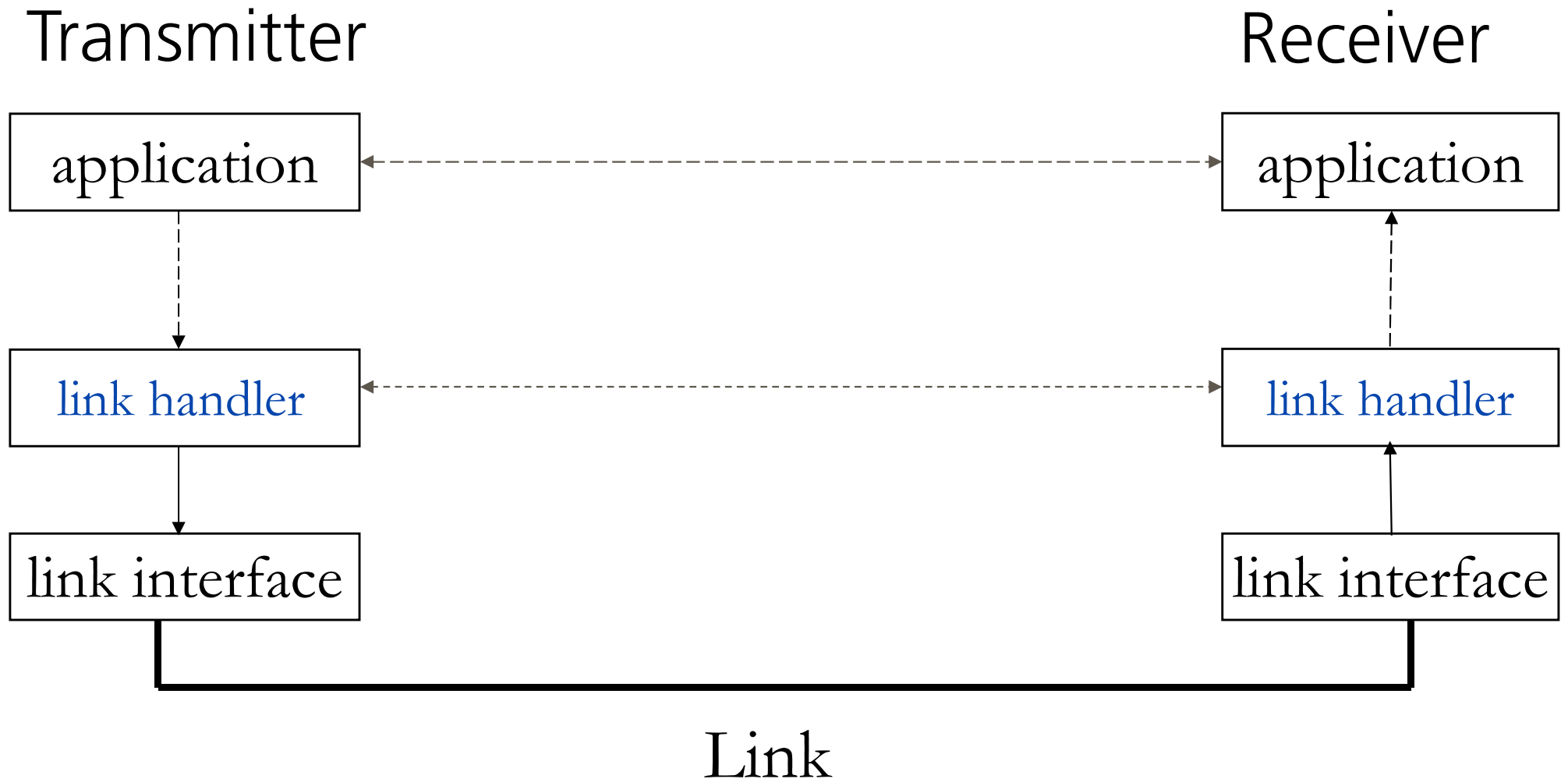
Flag = 01111110

16 or 32 bits CRC

Go-back-N or Selective-repeat ARQ

Protocol structure so far (1)

Link handler: Framing, error detection(?), error control(?)



Link/Channel User Modes

- Simplex
 - ◆ Signals possible only in one direction
 - Broadcast Radio/TV
- Half Duplex
 - ◆ Signals possible in both directions but only one at a time
 - Ch 16 VHF, Comm radio
- Full Duplex
 - ◆ Signals possible in both directions simultaneously
 - VHF traffic channels, Full duplex Ethernet
 - Two half duplex channels
 - POTS analog links

Performance: data rate and reach

- 10BASE5: 10 Mb/s, max 500 m
- 1000BASE-T: 1000 Mb/s, max 100 m
- 1000BASE-LX10: 1000Mb/s, max 10 km (SM)
- ADSL2+: 24 Mb/s downstream,
reach <5km
- VDSL2: 50 Mb/s downstream,
reach <500m
- WiFi 802.11n: >72 Mb/s (MIMO),
reach indoor ~70m, outdoor ~250m
- 4G: 100Mb/s (mobile) 1Gb/s (stationary)

Reach Limitations

- Dampening
- Noise
- Cross talk/Interference
- Dispersion
 - ◆ Intermodal: Modes take different path
 - ◆ Wavelength: Wavelengths have different propagation speed
- Enter: Repeater!
 - ◆ Regenerates signal

Protocol structure so far (2)

