Data Communication and Computer Network

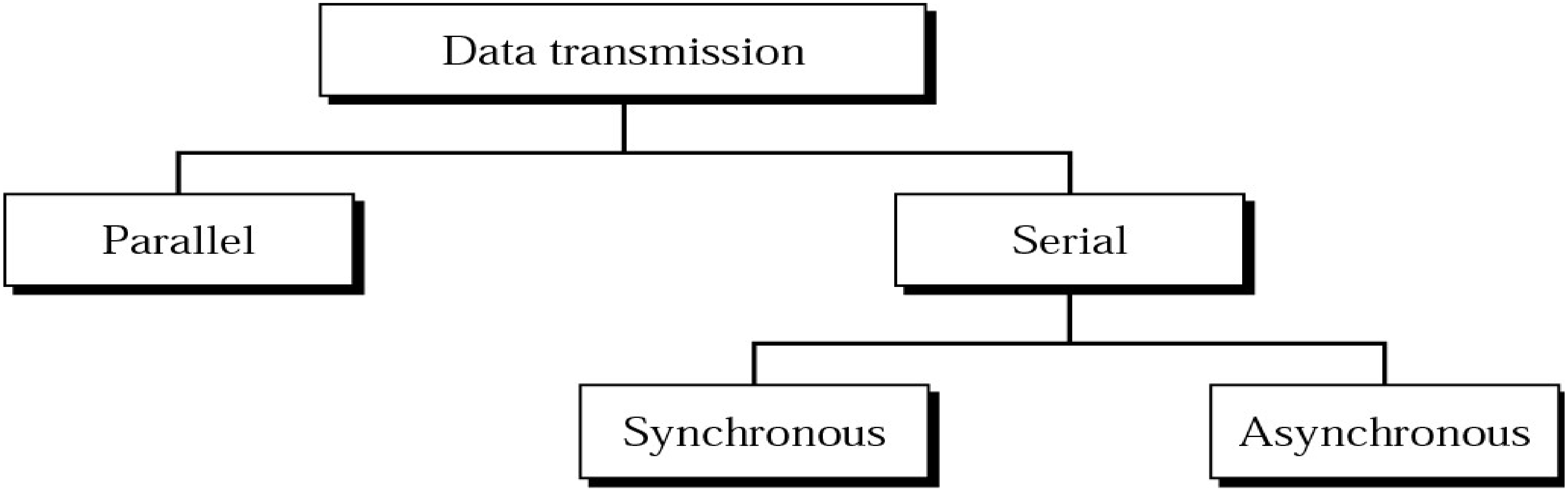
Data Link Layer

Responsibilities

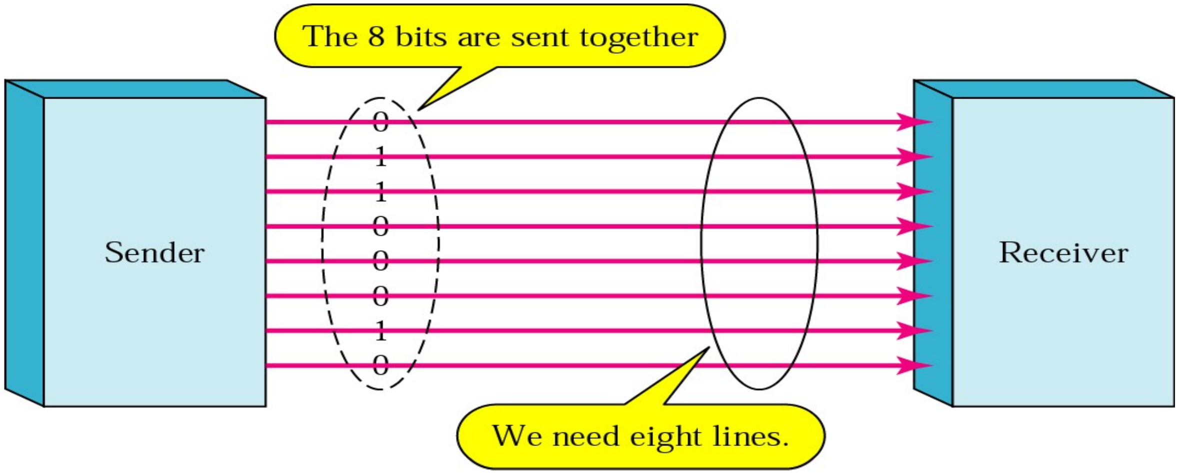
- Framing
- Error Detection
- Error Control
- Flow Control
- Medium Access Control

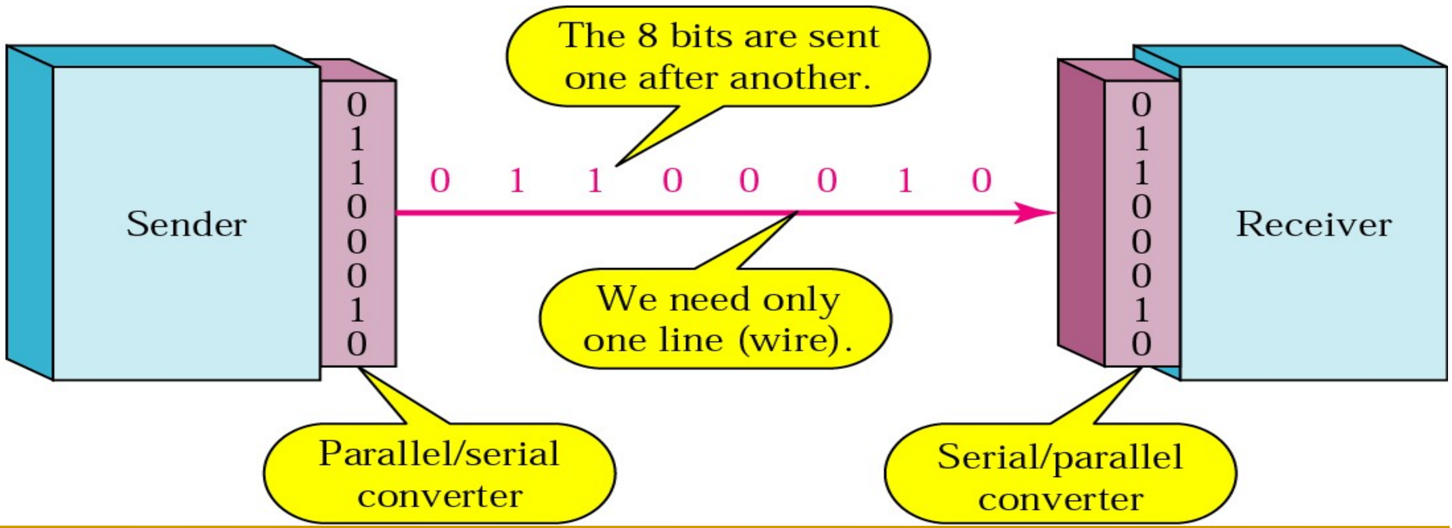
Framing

Mode of Transmission



Parallel and Serial Transmission





Framing

- Tx sending a sequence of bits to Rx over a link
  - Bits encoded into signal
  - Signal sent as series of voltage levels

- Framing: break up long bit-sequence into multiple relatively small ‘frames’ before send
  - Fixed / variable sized blocks of bits
  - Why framing?

✓ Needs identifier in each frame to distinguish between frames: frame sequence number

Frame synchronization

- What does the receiver need to know in order to read a frame correctly?
  - Where a frame ends or begins
  - At what intervals to sample the link to read the bits?
- Two common approaches
  - Asynchronous transmission: Rx and Tx agree on a pre-defined data rate
  - Synchronous transmission: Rx does not know the data rate being used by Tx

Asynchronous transmission

- Tx and Rx both know data rate, frame format
- Tx, Rx have separate clocks, no effort made to synchronize them

- Data transmitted one character at a time

  <start bit> 5-8 bits data <stop bit>

- Leading edge of start bit starts Rx clock
  - Once Rx clock starts, it runs at the pre-defined rate known to Rx (expected to be same as in Tx)
- How to tackle clock drift?

Synchronous transmission

- Rx may not know the data rate to be used by Tx

- Before starting to send data, Tx sends a known bit pattern (**fill pattern**)
  - At the same data rate at which it will transfer data
  - For a sufficiently long period of time
  - Rx adjusts own clock so that it can read the known fill pattern properly
- For Rx to detect start/end of data frame

  preamble bit pattern, ***post-amble*** bit pattern
- Fill pattern, pre/post-amble specified by protocol

Synchronous transmission (contd.)

- Desired: data block may be arbitrarily long

- After sending preamble, data transmission begins
  - To counter clock drift, Tx sends clock signals along with data
  - Separate clock signal, or clock signal embedded in encoded data signal e.g. Manchester encoding

- ✓ Pro: less overhead: few bits for large data block
- ✓ Con: clock signals need to be sent for some time before actual data is sent

Synchronous transmission (contd.)

- What if fill pattern / preamble / postamble is part of data? Use bit-stuffing

Example of synchronous protocol: HDLC

Preamble = post-amble = inter-frame fill = 01111110

- What if 01111110 is part of data?
  - ✓ whenever five consecutive 1's seen in data, Tx inserts an extra 0 after the five 1's

* HDLC : High-Level Data Link Control

Bit Stuffing

- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag

Original pattern:

111111111111011111101111110

After bit-stuffing:

1111101111101101111101011111010

Error Detection

Errors

- Data can be corrupted during transmission
- For reliable communication, errors must be detected and corrected
- Types of error
  - Single bit error
  - Burst error

Error detection

- Rx receives a signal, it samples and decodes signal to get binary data
- Error detection: how does Rx know if the data is actually what Tx sent?
- Use redundancy: extra bits added to data bits
- *Error correction*
  - Rx can detect whether errors are present and also correct the errors (can find which bits are in error)
  - Larger number of extra bits required than for error detection
  - Not commonly used, because of large overhead

Parity check

- Tx: one extra 'parity' bit added to each data unit
  - Odd parity: bit added so as to make # of 1's odd
  - Even parity: bit added to make total # of 1's even
- Rx: counts total number of 1's in the data unit, including the parity bit
- Detects any odd number of bit errors, but can be fooled by any even number of errors

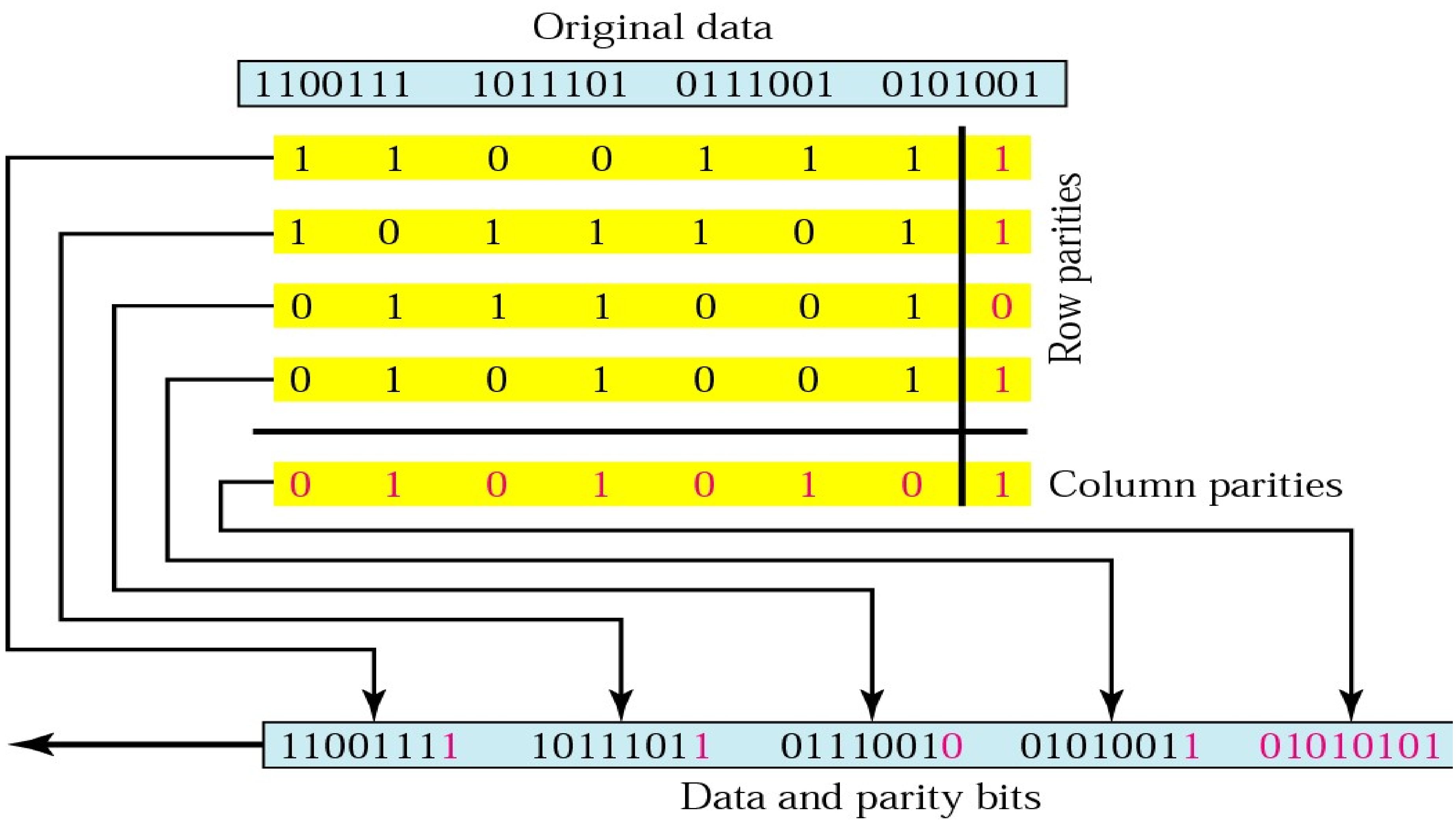
✓ Simple, easy to implement

✓ Not very robust against noise

Two-Dimensional Parity

- A block of bits divided into (say) 7-bit units (row)
- One parity bit computed for each row
- Also, parity bits computed considering the i-th bit in each row as a sequence (column), for all values of i = 0, 1, …, 7
- A redundant row of parity bits (column parity bits) added to the whole block

Two-Dimensional Even Parity



Cyclic Redundancy Check (CRC)

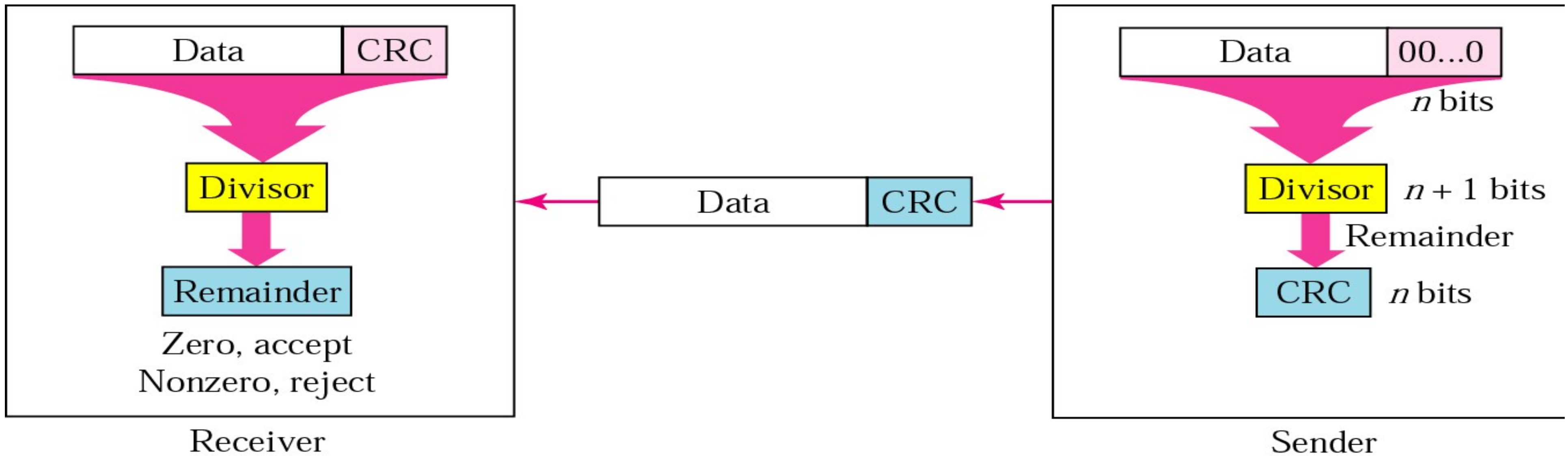
- Much more powerful method, easy to implement
- D: d-bit data
- R: r-bit error detecting code (appended to D)
  - Often called Frame Check Sequence (FCS)
- T: (d+r)-bit frame to be transmitted
- P: (r+1)-bit pattern
- Value of r and pattern P known to Tx, Rx
- Modulo-2 arithmetic
  - Addition, subtraction of bits both implemented as XOR with no carry, no borrow
  - $0 \pm 0 = 0; 0 \pm 1 = 1; 1 \pm 0 = 1; 1 \pm 1 = 0$

CRC – the method

- At transmitter
  1. Extend D with r 0's to the right (less significant bits)
  2. Divide extended D by P (of (r+1)-bits) using mod-2 arithmetic, to get remainder R (of r bits)
  3. Append R to the right of M to get T (d+r bits)
  4. T is transmitted

- At receiver
  1. Divide received T by P, using mod-2 arithmetic
  2. If remainder not zero, then error

CRC – the method



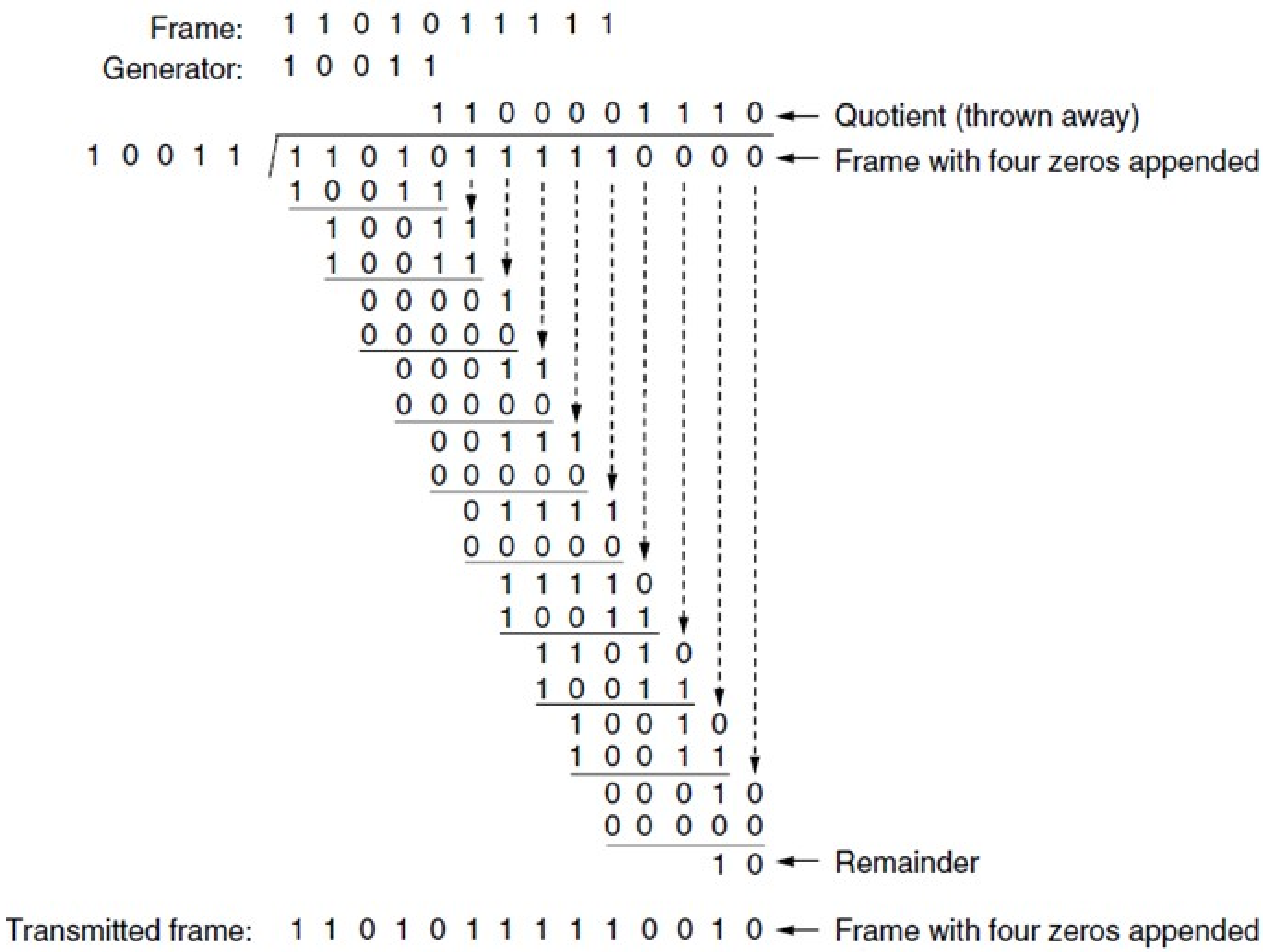
CRC – an example

Message D = 10101, d = 5

Let the divisor be fixed as P = 1101 (r+1 bits), so r = 3 redundant bits will be appended to data

What is the mathematical logic in CRC?

An example of CRC calculation



CRC in terms of polynomials

- Any bit pattern can be expressed as a polynomial in (a dummy variable) x, containing the powers of x corresponding to the '1' bits

  $110011 \equiv x^5 + x^4 + x^1 + x^0$

- Commonly used divisors P

  CRC-16 = $x^{16} + x^{15} + x^2 + 1$

  CRC-CCITT = $x^{16} + x^{12} + x^5 + 1$

  CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^{12} + x^4 + x^2 + x + 1$

What errors can CRC detect?

- ✓ All single-bit errors
- ✓ All double-bit errors, as long as long as P has at <u>least three 1s</u>
- ✓ Any burst error for which the length of the burst is less than or equal to the length of the FCS (frame check sequence)
- ✓ Many other larger burst errors

- ✓ But, still NOT fool-proof

  Errors may occur and may not be detected by CRC

Error Control

Error control

Rx can detect if there is error in the received frame, but if there is, then what?

- Error control
  - Ensures that the finally received bit pattern is same as the sent bit pattern
- Forward error control
  - Error recovery by correction at the receiver
  - Requires large number of error correcting bits to be added to data (e.g. Hamming code)
- Backward error control
  - Error recovery by retransmission: Automatic Repeat Request (**ARQ**)

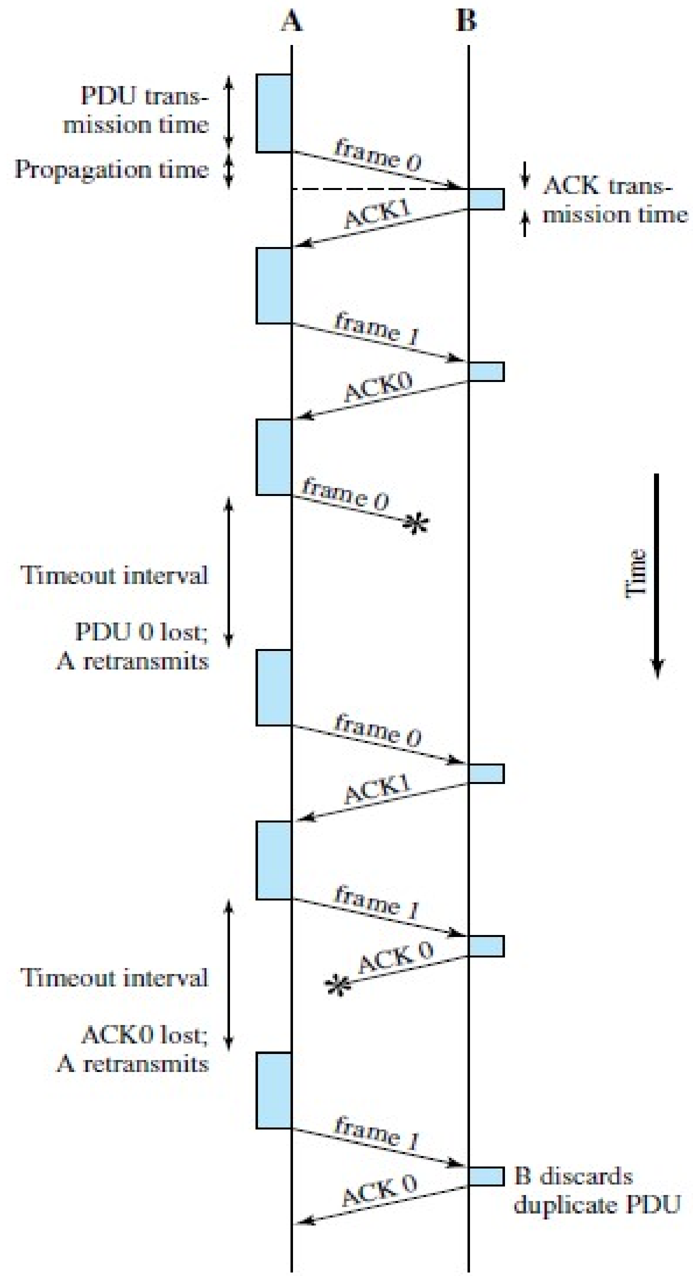
Stop & Wait ARQ

- 1 bit sequence number in each frame (0 or 1)

- Tx: send a frame, wait for ACK / NAK from Rx
- Rx: receive frame, check, send ACK / NAK
  - ACK includes number of next frame expected (0 or 1)
- Tx:
  - Get ACK => send next frame
  - Get NAK => re-transmit previous frame

- Simple and minimum buffer requirement

Stop & Wait ARQ
An Example



Stop & Wait ARQ - complexities

- Frames / ACKs can get lost => use timeout
- Transmitter Tx
  - send a frame, wait for ACK from Rx
  - If get ACK, send next frame
  - If no ACK within a timeout period (or get NAK), re-send previous frame
- Receiver Rx
  - check received frame for errors, send ACK if frame is ok
  - If error in frame, discard the frame and do NOT send ACK (or, may send NAK – time vs message tradeoff)
- Duplicate frames? Duplicate ACKs?

Efficiency of ARQ scheme

K = size of data frame in bits

D = data rate of channel in bits/sec

L = length of channel in meters

V = propagation speed in channel in m/sec

S = size of ACK frame in bits

D is the raw data rate

- at what rate can bits be put onto the channel

Effective data rate E

- At what rate is useful data being transmitted

Channel utilization: E / D

Efficiency of ARQ scheme (contd.)

- Channel utilization = 1 / (1 + 2 (L/V) / (K/D))

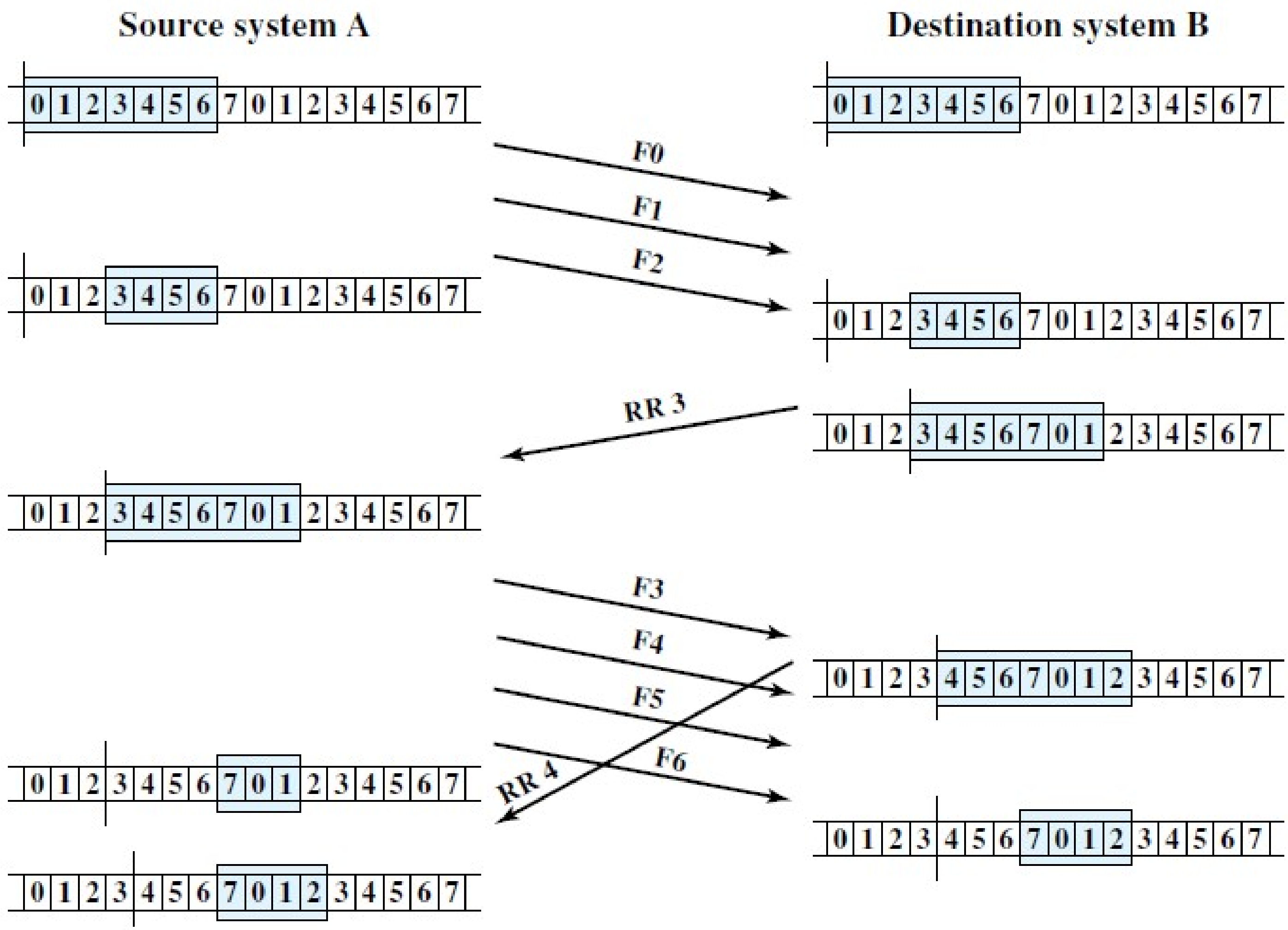
- So, Stop & Wait ARQ is NOT efficient if
  - Long links (high propagation time)
  - High data rate (low transmission time)
  - Frame size is very small (frame transmission time << propagation time)

- If message size (K) increased, utilization increases, but more buffer space required

Go-Back-N ARQ

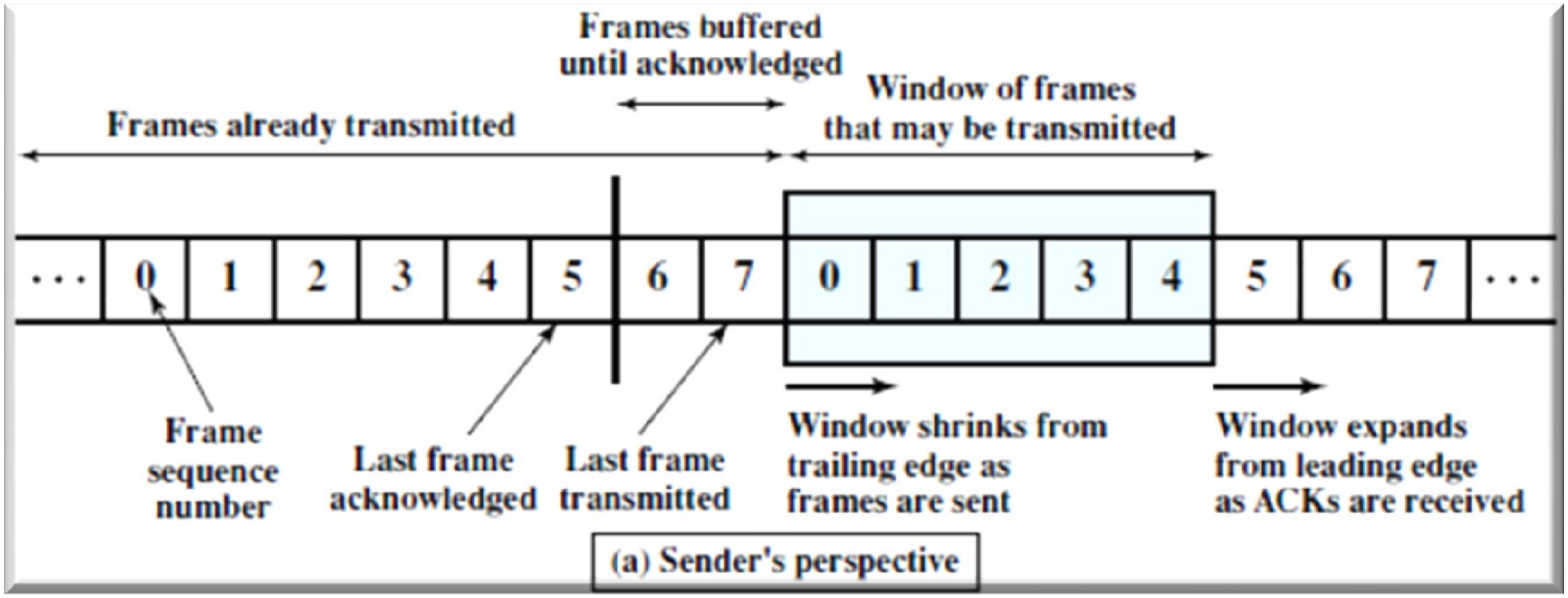
- Tx has a sequence of frames to transmit
- Frame seq. no: assume 3 bits (0,1,…,7,0,1,…)
- Tx allowed to send W (> 1) frames before waiting for ACK
- Window of frames: number of frames that Tx is allowed to send without receiving any ACK
  - W=4 implies Tx can send frames 0, 1, 2, 3 without waiting for any ACK from Rx
- Rx allowed to receive frames in order
- ACKs can be cumulative

Sliding-Window concept

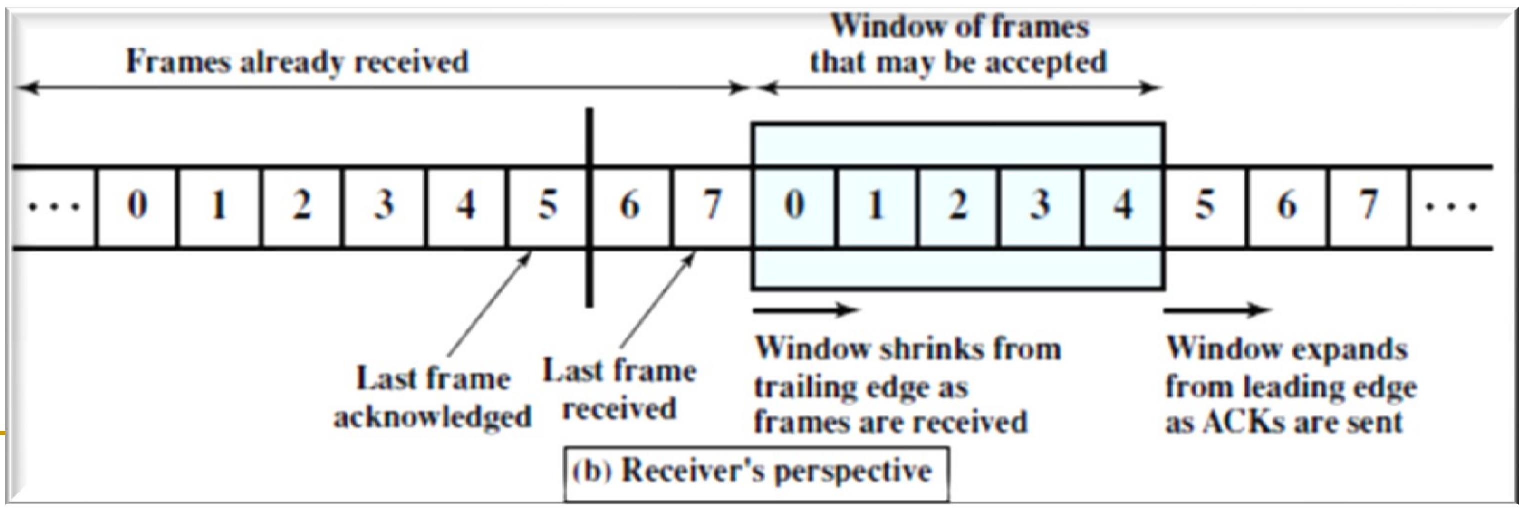


ACK or RR (Receive Ready) N : "***I have received all frames up through frame number N-1 and am ready to receive frame number N; in fact, I am prepared to receive W frames, beginning with frame number N***"

Sliding-Window concept Cont’d



(a) Sender's perspective



(b) Receiver's perspective

If error detected in a frame

- Rx detects error in a received frame
  - May send a NAK giving sequence no. of that frame, or may send nothing
  - Discard this and all future frames until the frame in error is correctly received (in-order receipt of frames)

- When Tx gets a NAK for a frame, or does not get ACK for a frame within the timeout period
  - Re-transmit this frame as well as all subsequent frames

- Tx has to remember each unacknowledged frame (at most W frames)

Lost / damaged data frame F_i

- Case 1: Tx has more frames to send
  - Tx sends F_{i+1}, F_{i+2}, … up to window limit
  - If Rx receives any of these frames, Rx discards them and can send NAKi
  - Tx gets NAKi before timeout, re-sends F_i, F_{i+1}, …

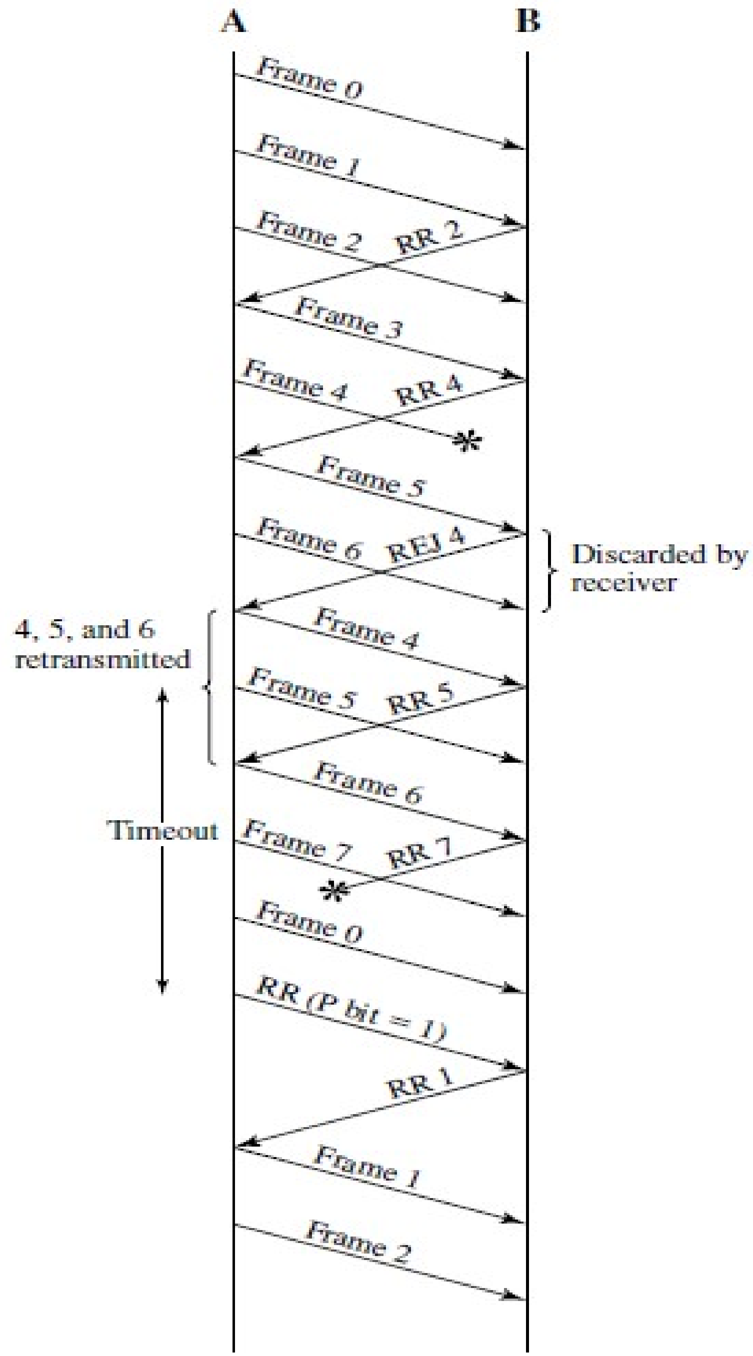
- Case 2: Tx times out
  - Pessimistic approach: Tx re-sends F_i, F_{i+1}, F_{i+2} …
  - Optimistic approach: Tx sends a poll message RR to ask Rx what frame it expects next, Rx replies with NAKj
  - Tx re-transmits frames F_j, F_{j+1}, F_{j+2} …

Lost / damaged ACK for frame F_i

- Case 1: Tx gets ACK for frame F_j, F_j sent later than F_i
  - Since ACKs are cumulative, this is implicitly an ACK for frame F_i also, Tx simply extends window
- Case 2: Tx times out
  - Tx does not know whether data frame got lost or whether data reached Rx and ACK got lost
  - Similar to Case 2 for lost data frame

Go-Back-N: An example

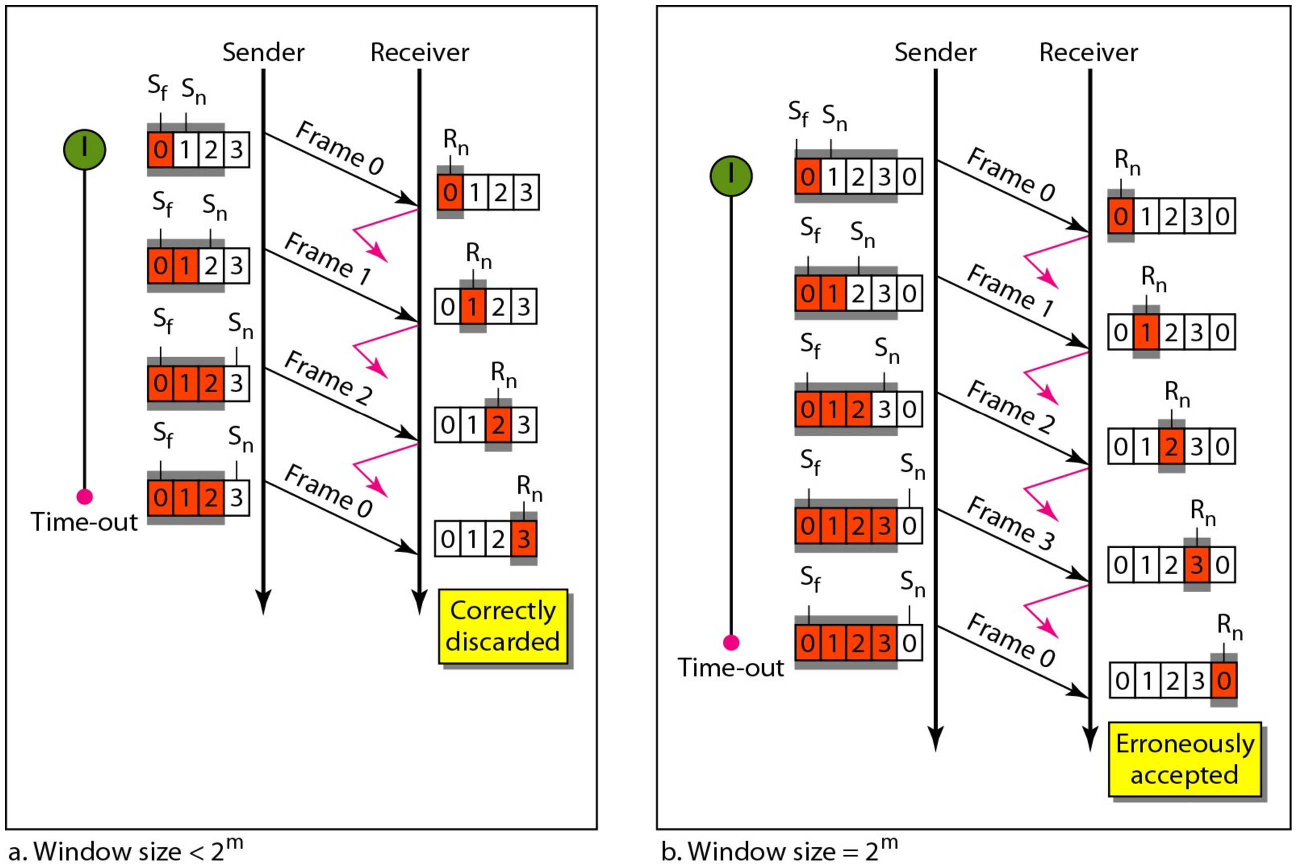
- 3 bit sequence number
- ACKs indicated as RR
- NAKs indicated as REJs



❖In this diagram, Optimistic approach is assumed
❖**RR (Receive Ready) is same as ACK**
❖**REJ (Reject) is same as NACK**

Go-back-N : Relationship of Sequence Number and Window Size

For k-bit sequence numbers, maximum window size for Go-Back-N ARQ is $(2^k - 1)$



In this diagram, Pessimistic approach is assumed

Selective-Reject/Repeat ARQ

- Allow Rx to receive frames out of order

  If F_i contains errors, Rx discards F_i and sends NAKi

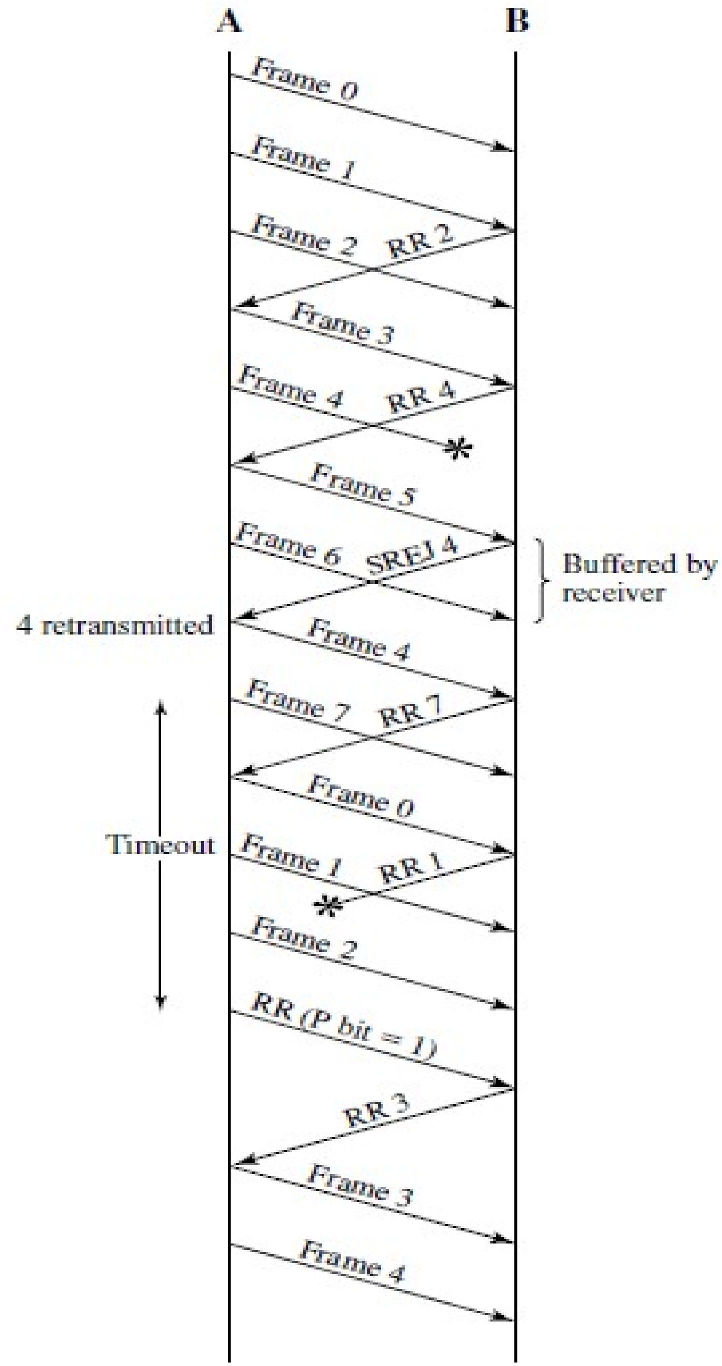
  - Rx accepts F_{i+1}, F_{i+2}, … (if they are error-free) even if F_i has not been received properly

- ✓ Motivation: reduce number of re-transmissions

- However, applications at higher layers usually require in-order frames
  - Rx must buffer frames being accepted out of order

Selective-Reject/Repeat ARQ
An example

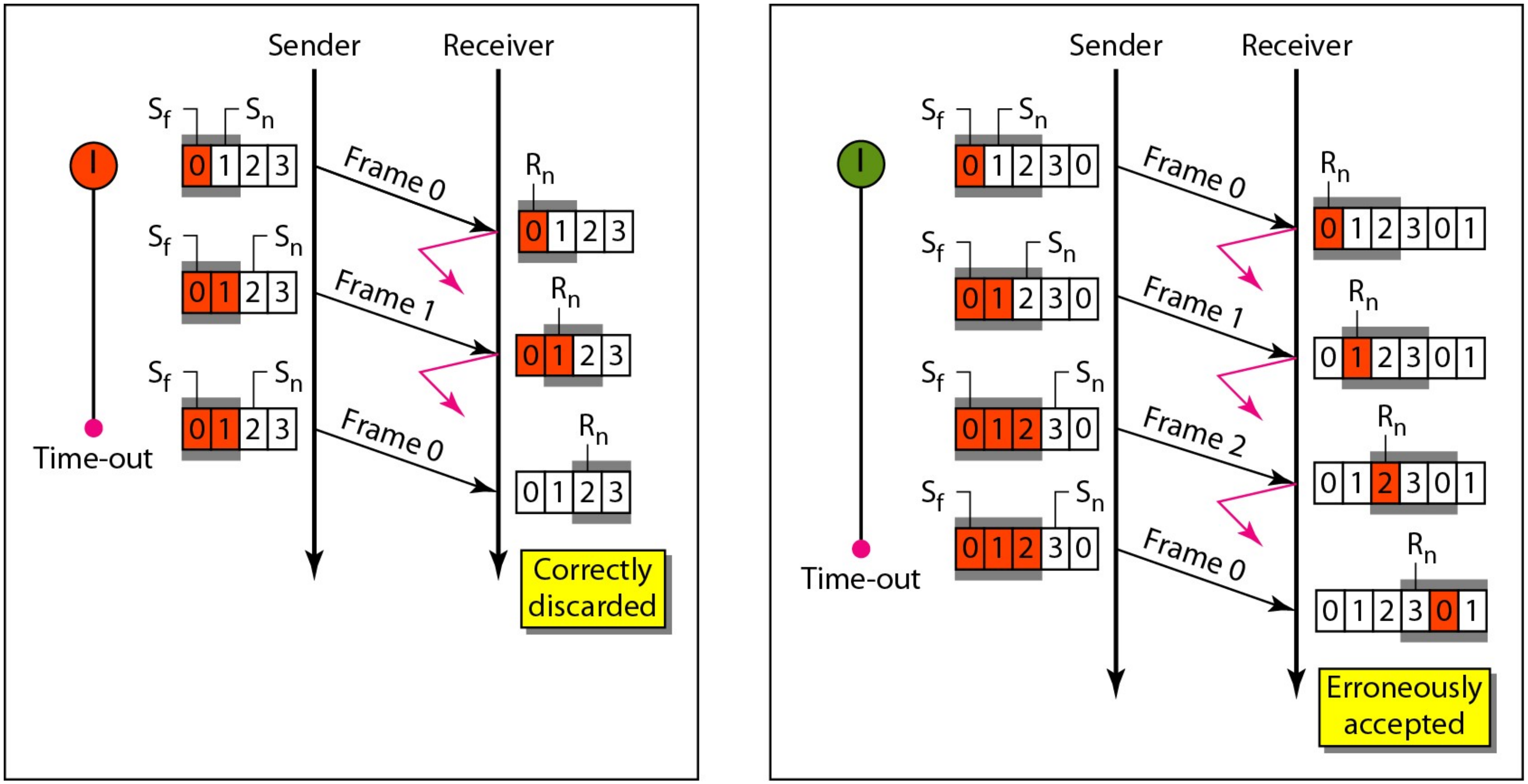
- 3 bit sequence number
- ACKs indicated as RR
- NAKs indicated as SREJs



❖In this diagram, Optimistic approach is assumed
❖**RR (Receive Ready) is same as ACK**
❖**REJ (Reject) is same as NACK**

Selective Reject/Repeat : Relationship of Sequence Number and Window Size

For k-bit sequence numbers, maximum window size for Go-Back-N ARQ is 2^{k-1}



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

In this diagram, Pessimistic approach is assumed

Sliding Window scheme enhancements

- Rx can acknowledge frames without permitting further transmission (Receive Not Ready)
  - When Rx becomes ready, must send a message to allow Tx to resume sending frames
- Piggybacking
  - Often both sides transmit and receive data: each station maintains a sender window and a receive window
  - Suppose A sends a data frame to B, ACK for this frame can be piggybacked on a data frame from B to A
  - Frame format includes a field to hold a sequence number (of this frame) plus a field to hold the sequence number used for ACK

Flow Control

Stop & Wait Flow Control

Sliding Window Flow Control

Flow Control

- What if Tx sends frames too fast for Rx to handle?
  - Rx may not be able to process the data as fast as Tx is sending it
  - Rx can buffer, but buffer has finite size. Once that is filled, data will be lost

- Flow control: technique to control data flow between sender and receiver so that sender is blocked if receiver cannot accept any more data

Stop & Wait Flow Control

- Tx: send a frame, wait for ACK from Rx
- Rx: receive frame, check for errors, send ACK when ready for another frame
- Similar to Stop & Wait ARQ, only difference in the time to send ACK
  - Error control: Rx sends ACK immediately on understanding that there is no error
  - Flow control: ACK delayed till Rx processes the received frame in some way
- Simple, low buffer requirements at both Tx and Rx, but low line utilization

Sliding Window Flow Control

- Similar to Go-Back-N
- Rx can allocate buffer space for W frames
  - So, window size W use
  - Tx maintains sender window: a list of sequence numbers that it is allowed to send without waiting for an ACK
  - Rx maintains receiver window: list of sequence numbers that it is prepared to receive
- Rx sends ‘ACK N’ only when
  - it has received all frames up to N-1 correctly, and
  - it is ready to receive W more frames starting from frame N

References

- *Data Communications & Networking, 5th Edition, Behrouz A. Forouzan*
- *Computer Networks, Andrew S. Tanenbaum and David J. Wetherall*
- *Wikipedia*