

# Reliable Data Transfer between Two Machines

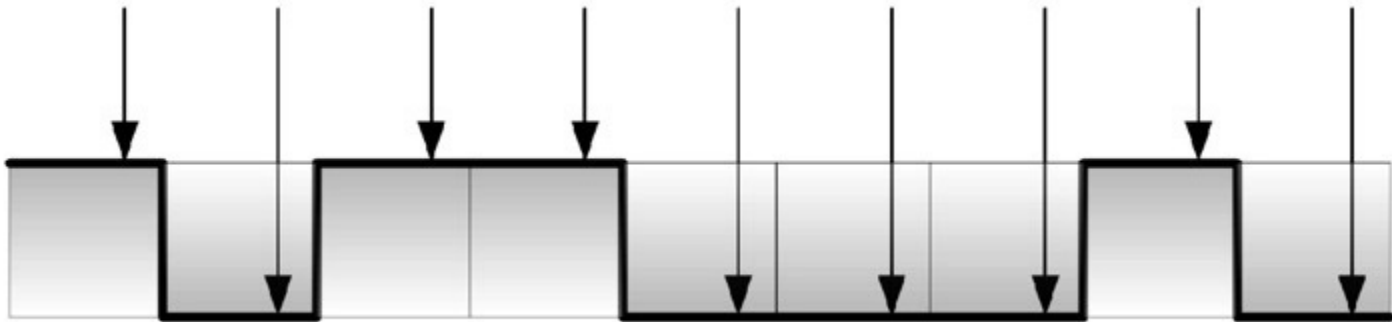
# Issues

- Synchronization
  - How does the receiver know when the sender is sending data?
- Encoding
  - Representing data with signals
- Error control
  - How to ensure the receiver gets the correct data
- Flow control
  - How to ensure the sender does not swamp the receiver (fast sender, slow receiver)

# Synchronization

# Need for synchronization

- Basic steps in transmitting digital data
  - Transmitter sends a stream of 0 or 1 bits
  - Receiver samples incoming signal **once per bit time** to see if it is a 0 or 1
    - Typically at the center of the bit



- Transmitter sends one bit (0 or 1) every 1 millisecond (say)
  - Say there is a clock that ticks every 1 millisecond
  - Transmitter puts a 1 or 0 on the line at each tick of the clock
    - bit 1 on tick 1, bit 2 on tick 2, ...
- What should the receiver do to get the pattern correctly?
  - Need to know when to start counting (when will bit 1 arrive)
    - A-priori agreed or sender has to tell (how?)
  - Need to know when to look for the next successive bits (duration of one bit)

# Framing

- Break up the bit pattern into multiple frames
- Issues
  - Needs identifier for each frame to distinguish between frames
  - Needs technique to identify where a frame ends and the next one begins
    - Frame synchronization problem

# Frame Synchronization

- When data is transferred from the transmitter to the receiver, unless steps are taken to provide synchronization, the receiver may start interpreting the data erroneously
- Two common approaches:
  - Asynchronous Transmission
  - Synchronous Transmission

# Asynchronous Transmission

- Transmitter and Receiver has separate clocks
- Data rate and frame format negotiated a-priori
- Data are transmitted one character at a time (5-8 bits)
  - Timing or synchronization must only be maintained within each character
  - Start bit at the beginning of each new character to resynchronize (start the receiver clock at the right time)
- When no character is being transmitted, the line between transmitter and receiver is in an *idle* state
  - Start, stop, and parity bits.



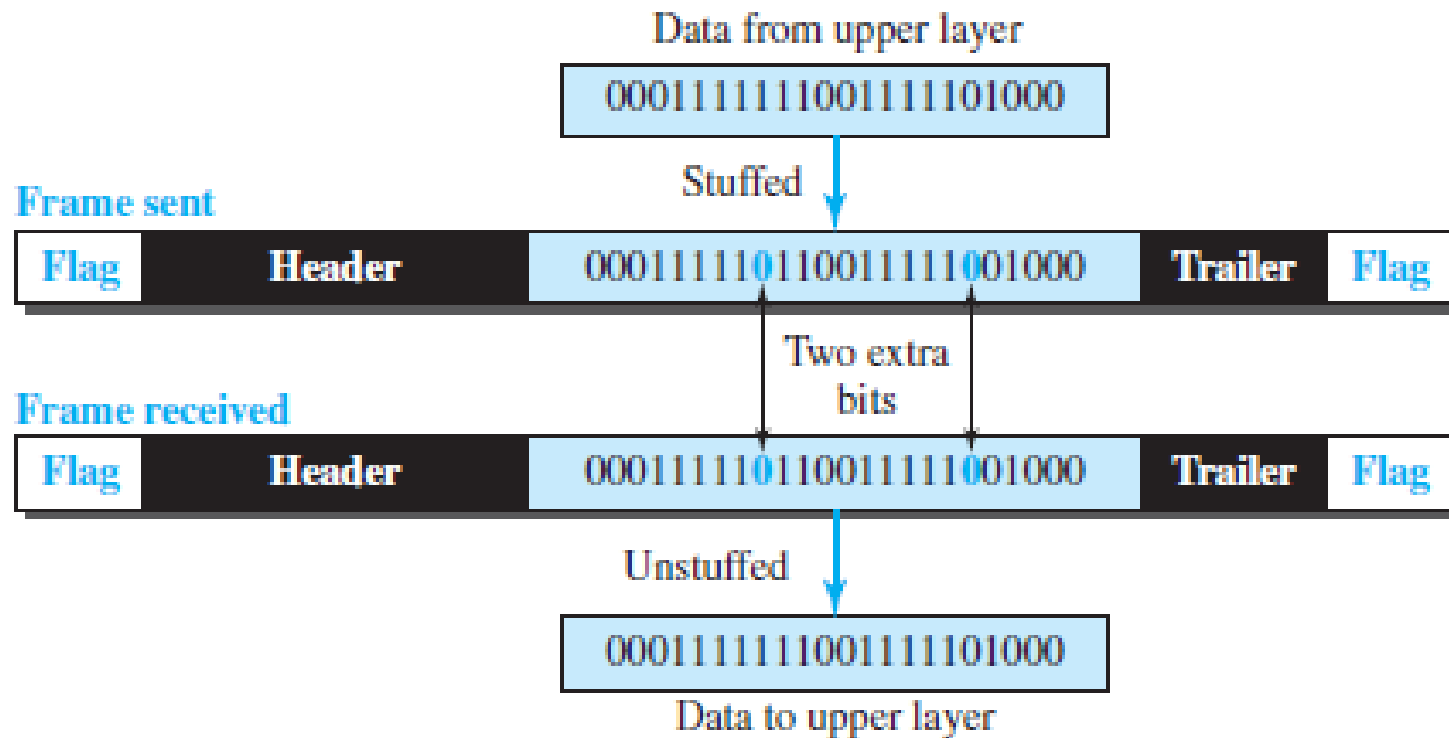
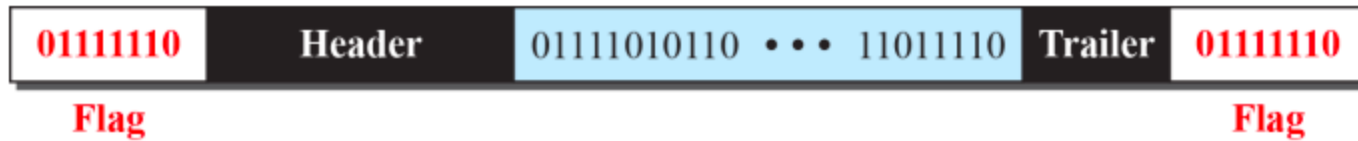


# Synchronous Transmission

- Receiver clock is synchronized with transmitter clock
- A block of bits is transmitted in a steady stream without start and stop codes
  - The block may be arbitrarily long
  - To prevent timing drift between transmitter and receiver, clock signal is embedded in the data signal or separate clock signal sent
- There is another level of synchronization required, so as to allow the receiver to determine the beginning and end of a block of data
  - Every block begins with a **preamble** bit pattern, and generally ends with a **postamble** bit pattern

# Synchronous Transmission (contd.)

- A typical synchronous frame format:
  - 8-bit flag (preamble)
  - Control fields
  - Data field
  - Control fields
  - 8-bit flag (postamble)
- Special inter-frame fill pattern is sent when no frame to send to maintain synchronization
- What if data itself contains the preamble/postamble pattern?
  - Use [bit-stuffing](#)



- For sizable blocks of data, synchronous transmission is far more efficient than asynchronous mode
  - Asynchronous transmission requires 20% or more overhead
  - The control information, preamble and postamble in synchronous transmission are typically less than 100 bits

# Encoding Techniques

# Data Encoding Techniques

- Data is transmitted by propagation and processing of signals
- Encoding – specifying how data is represented by signals
  - Digital data over digital signal
  - Digital data over analog signal

# Encoding Digital Data with Digital Signals

- Digital signal
  - Uses discrete, discontinuous, voltage pulses
  - Each pulse is a signal element
  - Binary data is encoded into signal elements
- Data
  - Bit string of 0's and 1's
  - Each bit is present for a duration T (**bit interval**)
  - Data rate =  $1 / T$  bps

# Some Terminology

- Unipolar
  - All signal elements have same sign
- Polar
  - Two logic states represented by +ve/-ve voltage
- Duration or length of a bit
  - Time taken for transmitter to emit the bit
- Modulation rate
  - Rate at which the signal level changes
- Mark and Space
  - Binary 1 and Binary 0 respectively



# Issues in Encoding

- Signal Spectrum
  - Lack of high frequencies reduces required bandwidth
  - Lack of dc component allows ac coupling via transformer, providing isolation reducing interference
- Clocking issues
  - Synchronizing transmitter and receiver is essential
  - External clock is one way used for synchronization
  - Synchronizing mechanism based on signal is also used & preferred (over using an external clock)

- Error detection
  - Can be built into signal encoding
- Signal interference and noise immunity
  - Some codes are better than others
- Cost and complexity
  - Higher signal rate (& thus data rate) lead to higher costs
  - Some codes require signal rate greater than data rate

# Some Encoding Schemes

- Nonreturn to Zero-Level (NRZ-L)
- Nonreturn to Zero Inverted (NRZI)
- Bipolar-AMI (Alternate Mark Inversion)
- Pseudoternary
- Manchester
- Differential Manchester

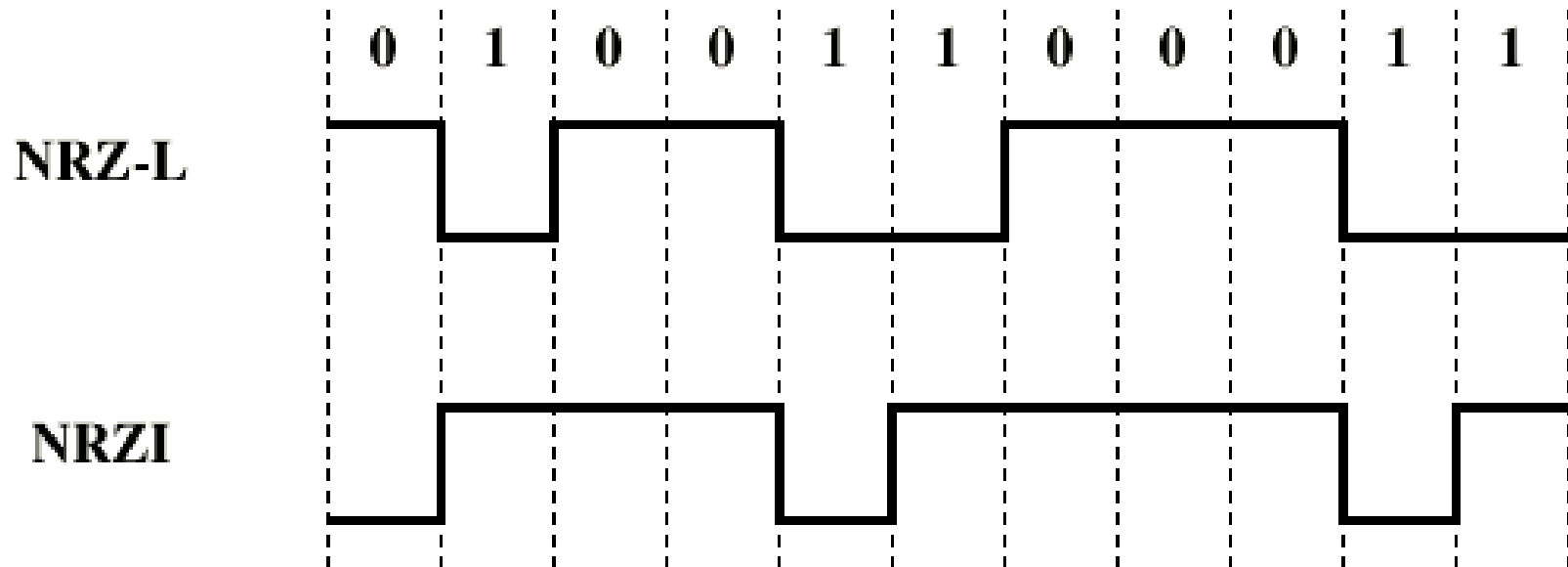
# Nonreturn to Zero-Level (NRZ-L)

- Two different voltages for 0 and 1 bits
- Voltage held constant during bit interval
- Example:
  - 0 = High
  - 1 = Low

# Nonreturn to Zero Inverted (NRZ-I)

- *Nonreturn to zero inverted on ones*
- Voltage held constant during bit interval
- Data encoded as presence or absence of signal transition at beginning of bit time
- Transition (low to high or high to low) denotes a binary 1
- No transition denotes binary 0
- An example of differential encoding (Data represented by changes rather than levels)

# NRZ



# NRZ pros and cons

- Pros
  - Easy to engineer
  - Makes good use of bandwidth
- Cons
  - Presence of DC component
  - Lack of synchronization capability, not self-clocking
    - Long string of 1's (or 0's) for NRZ-L or long string of 0's for NRZI will give constant voltage levels may cause loss of receiver synchronization

# Multilevel Binary

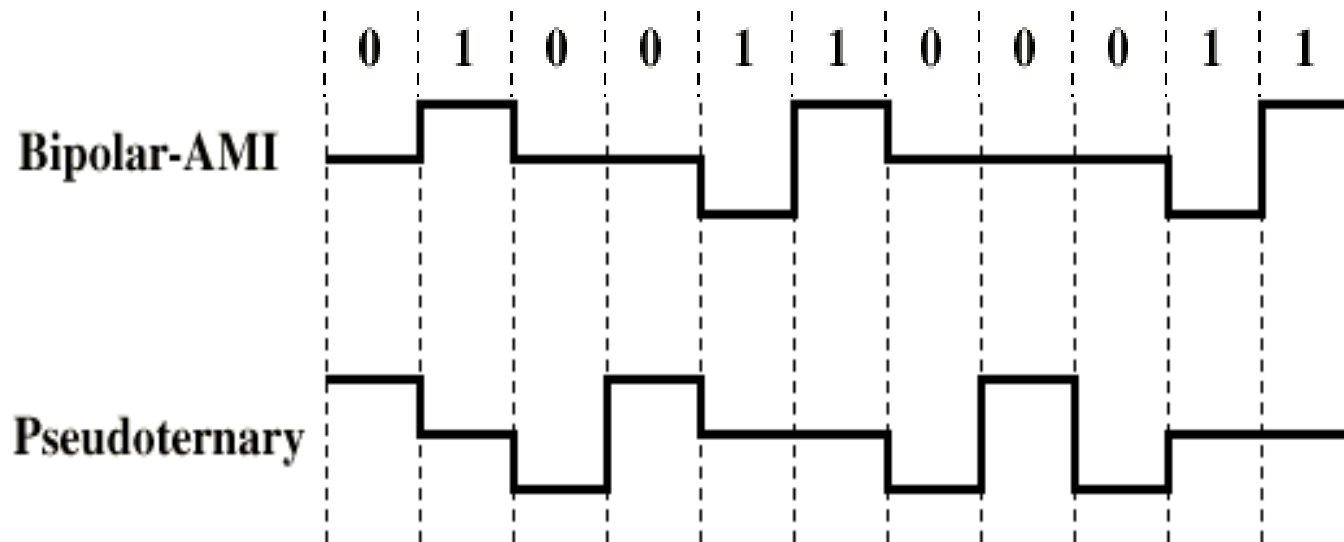
- Use more than two levels
- Bipolar-AMI
  - Bit 0 represented by no line signal
  - Bit 1 represented by positive or negative pulse
  - Pulses for bit 1 alternate in polarity
  - No loss of sync if a long string of 1's happens (long string of 0's still a problem)
  - No net dc component
  - Lower bandwidth
  - Some error detection builtin



# Pseudoternary

- Opposite of Bipolar-AMI
- Bit 1 represented by absence of line signal
- Bit 0 represented by alternating positive and negative
- No advantage or disadvantage over bipolar-AMI

# Bipolar-AMI and Pseudoternary



# Tradeoff for Multilevel Binary

- Not as efficient as NRZ
  - With multi-level binary coding, the line signal may take on one of 3 levels, but each signal element, which could represent  $\log_2 3 = 1.58$  bits of information, bears only one bit of information
  - Receiver must distinguish between three levels (+A, -A, 0), increased complexity
  - Requires more signal power for same probability of bit error

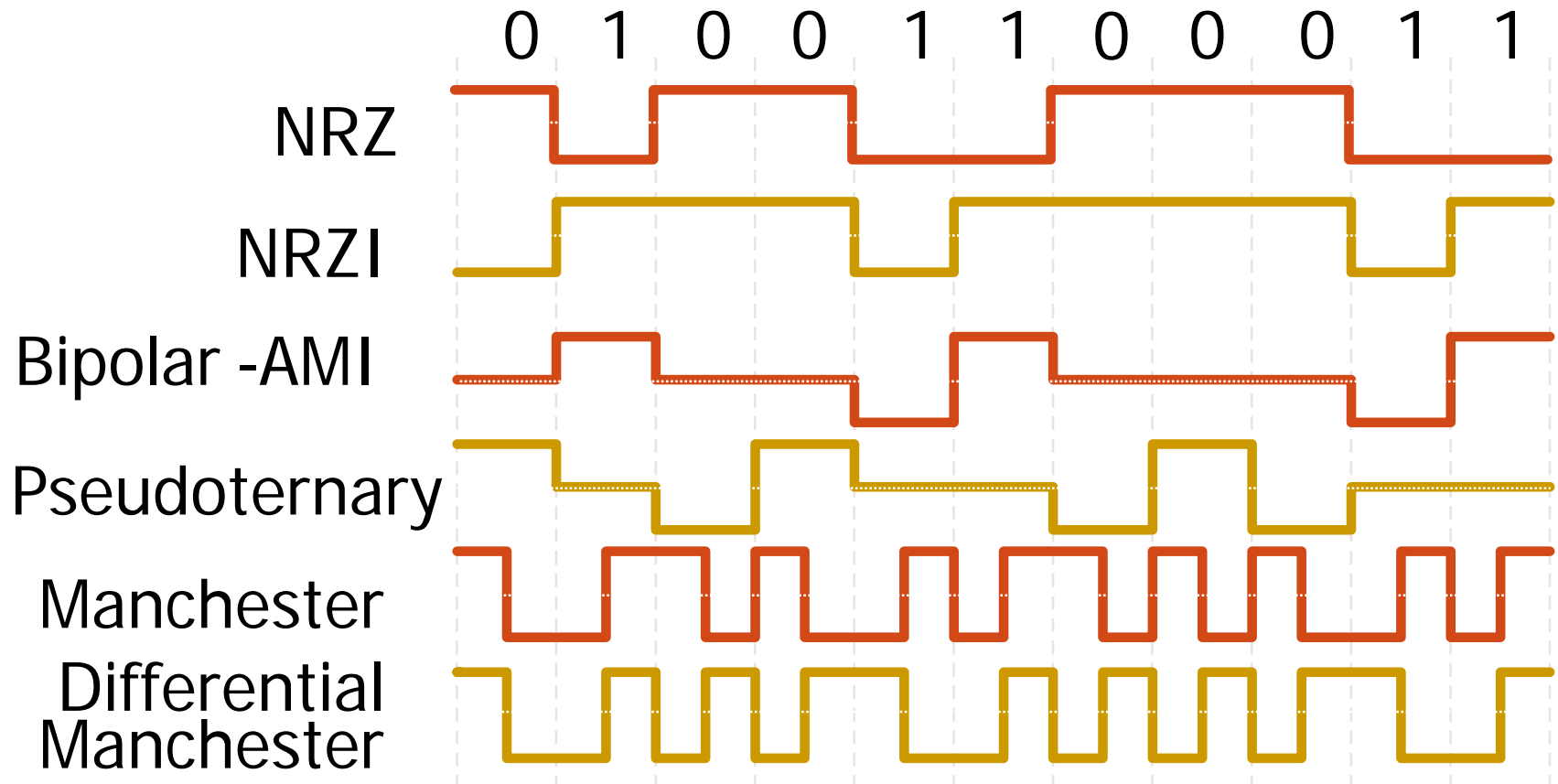
# Biphase

- Manchester
  - Transition in middle of each bit period
  - Transition serves as clock and data
  - Low to high represents 1
  - High to low represents 0
- Differential Manchester
  - Midbit transition is for clocking only (always there)
  - Transition at start of a bit period represents 0
  - No transition at start of a bit period represents 1
  - Differential encoding scheme

# Biphase Pros and Cons

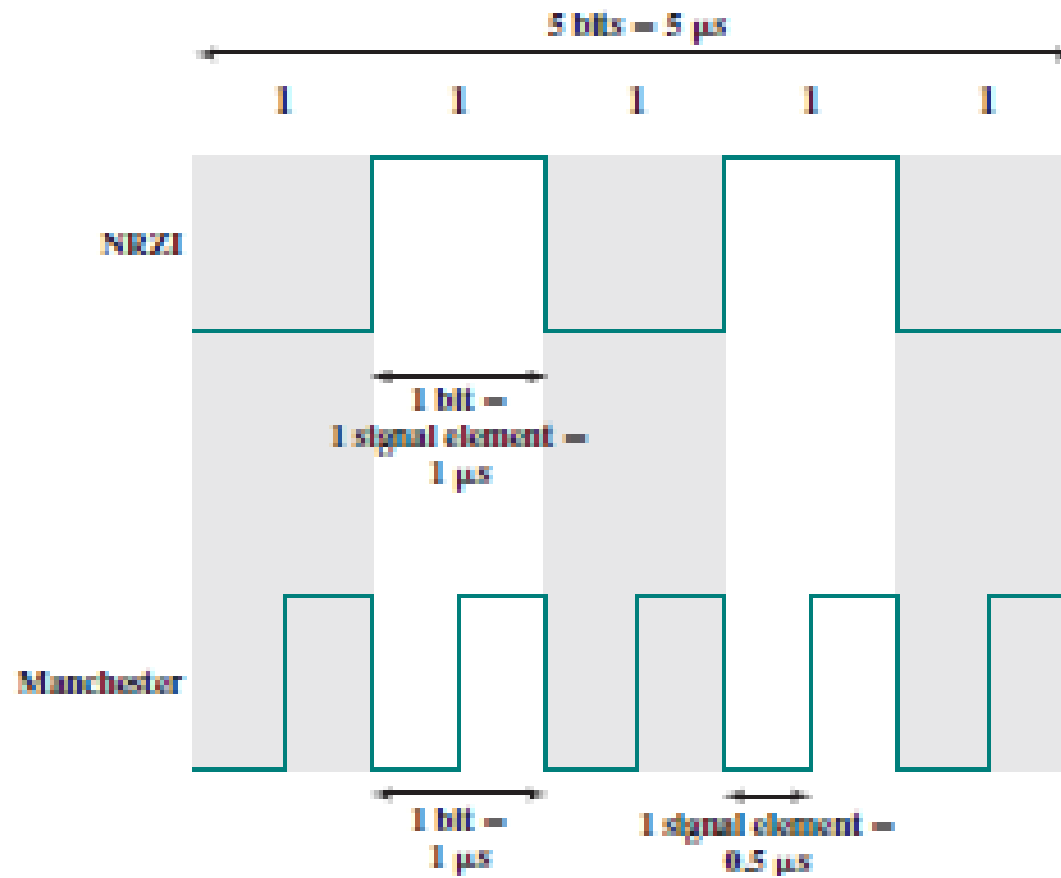
- Con
  - At least one transition per bit time and possibly two
  - Maximum modulation rate is twice that of NRZ
  - Requires more bandwidth
- Pros
  - Synchronization on mid bit transition (self clocking)
  - No dc component
  - Error detection
    - Absence of expected transition points to error in transmission

# Digital data, Digital signal



# Modulation Rate

- Modulation rate/ Signaling rate of an encoding scheme = rate of signal transitions in the channel
- Higher the modulation rate, higher the bandwidth required
- Depends on the actual data stream
  - Should see the maximum modulation rate possible for some data stream
- Another way to characterize : No. of signal transitions per data bit (average/maximum)





	Minimum	101010...	Maximum
NRZ-L	0 (all 0s or 1s)	1.0	1.0
NRZI	0 (all 0s)	0.5	1.0 (all 1s)
Bipolar-AMI	0 (all 0s)	1.0	1.0
Pseudoternary	0 (all 1s)	1.0	1.0
Manchester	1.0 (1010...)	1.0	2.0 (all 0s or 1s)
Differential Manchester	1.0 (all 1s)	1.5	2.0 (all 0s)

- So there is a tradeoff
  - Biphase is good for synchronization but needs high signaling rate (higher bandwidth)
  - Non-Biphase codes need lower signaling rate (so lower bandwidth) but may cause receiver to go out of sync
- Can we try to get good clock synchronization at low signaling rate?
  - Scrambling
  - $x\text{B}/y\text{B}$  codes (for different  $x, y$  values)

# Scrambling

- Replace sequences that will cause long spells of constant voltage with filling sequences that introduces sufficient transitions
- Receiver should replace filling sequences with original data sequence
- Example: Bipolar-AMI with 8-zeroes substitution (B8ZS)
  - For any octet of all 0's
    - If last voltage pulse before the octet is +ve, replace with 000+−0−+
    - If last voltage pulse before the octet is −ve, replace with 000−+0+−
  - How does the receiver detect scrambling has been done?
    - Hint: Is this a valid sequence for Bipolar-AMI?

# 8B/10B Encoding

- Other similar schemes like 4B/5B, 5B/6B, 64B/66B (Block Codes)
- Two levels of encoding
  - Bit pattern encoded with another bit pattern
  - Encoded bit pattern encoded again over digital/analog signals (as studied)
- Data split into 8-bit Octets
- Each octet is replaced with a predefined 10-bit code
- Each 10-bit code has one of the following
  - 5 0's and 5 1's
  - 6 0's and 4 1's
  - 4 0's and 6 1's
- Ensures that in the final encoded string, not more than 5 0's or 5 1's can occur simultaneously
  - Ensures enough transitions for clock synchronizations

- Also ensures DC-balance
  - In a string of at least 20 bits, difference in count of 0's and 1's is at most 2
  - This requires introducing *running disparity* bits (we will not cover)
- Requires less b/w, can go over longer distances
- Reduces data rate (as more no. of bits transmitted)
- Used in many protocols, Ex: Gigabit Ethernet

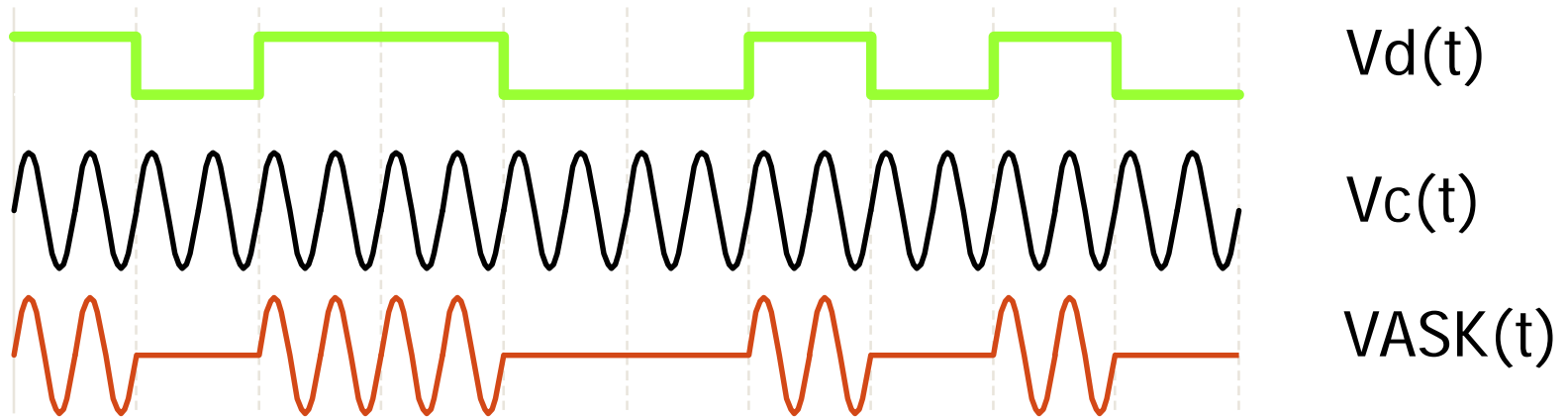
# Encoding Digital Data with Analog Signals

- Transmitting digital data through PSTN (Public telephone system)
  - 300Hz to 3400Hz bandwidth
  - modem (modulator-demodulator) is used to convert digital data to analog signal and vice versa
- Carrier signal: basic signal that is changed (modulated)
- Three basic modulation techniques are used:
  - Amplitude shift keying (ASK)
  - Frequency shift keying (FSK)
  - Phase shift keying (PSK)

# Amplitude Shift Keying

- Values represented by different amplitudes of a carrier signal
- Usually, one amplitude is zero
  - i.e. presence and absence of carrier is used
- Susceptible to noise
- Used over optical fiber (but usually with some digital encoding before ASK)

# ASK

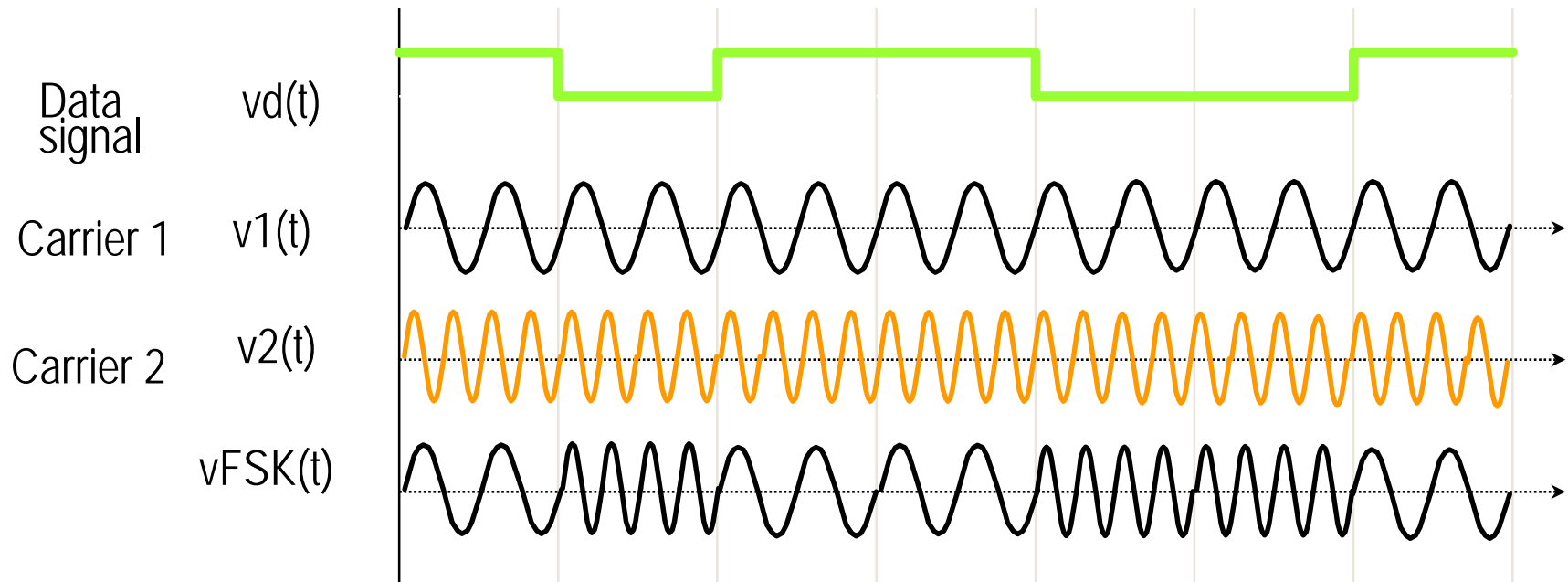




# Frequency Shift Keying

- Values represented by different frequencies (near carrier)
- Less susceptible to noise than ASK
- Up to 1200bps on voice grade lines, high frequency radio (3-30 MHz)
- Constrained by bandwidth of channel

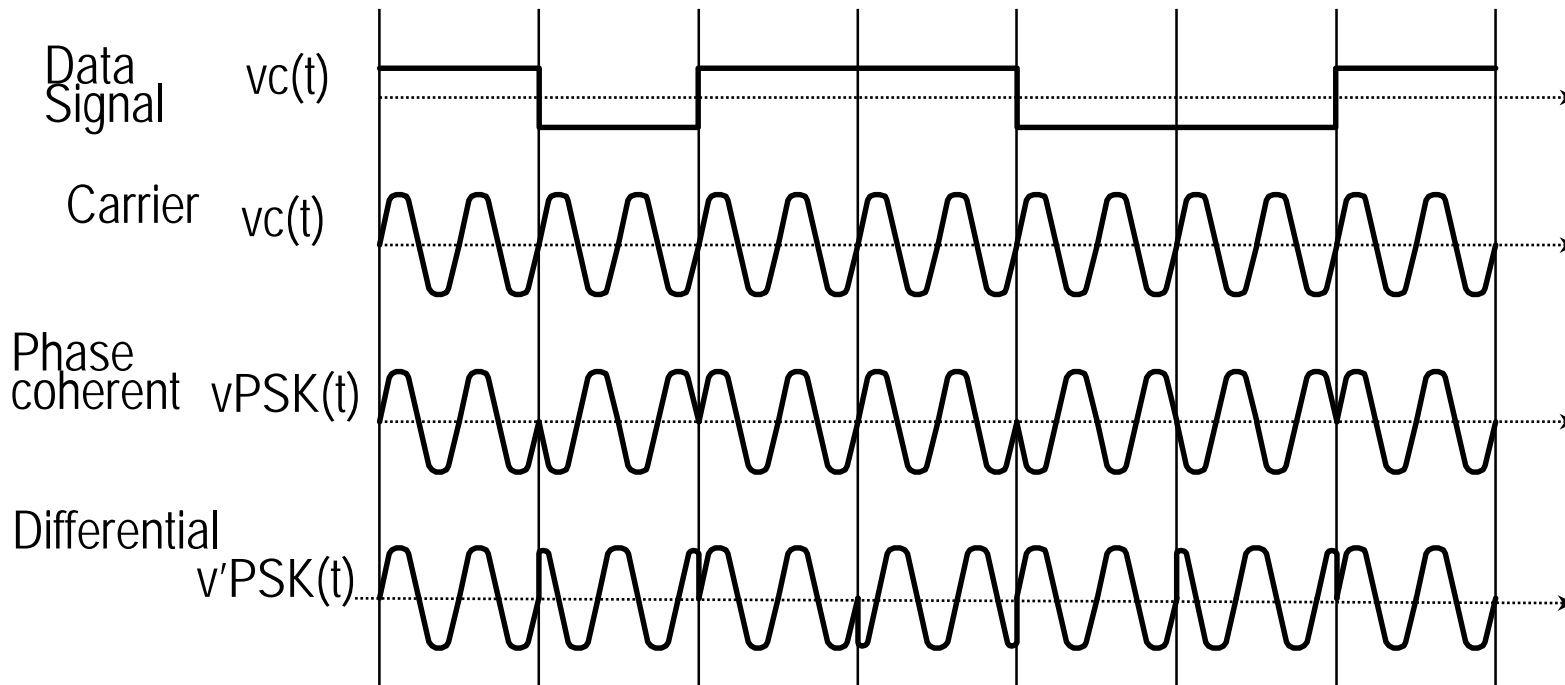
# FSK



# Phase Shift Keying

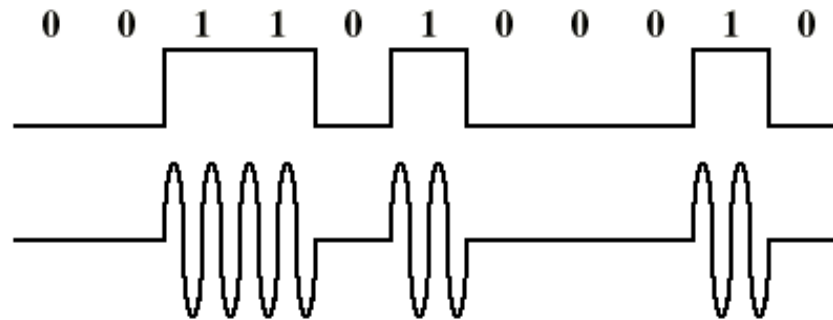
- Phase of carrier signal is shifted to represent data
- Differential PSK
  - Phase shifted relative to previous transmission rather than some reference signal

# PSK and DPSK



Differential example: for every logic 1,  
180 degree phase shift

# Modulation Techniques



(a) Amplitude-shift keying



(b) Frequency-shift keying



(c) Phase-shift keying

# Quadrature PSK

- More efficient use by each signal element representing more than one bit
  - e.g. shifts of  $\pi/2$  ( $90^\circ$ )
  - Each element represents two bits
  - Bit rate = 2 x baud rate
- High-speed modems, cellular base stations use different combination of phase and amplitude (*Quadrature Amplitude Modulation or QAM*)
  - 16-QAM (4 amplitude levels and 4 phase levels), allows one (amplitude, phase) combination to encode 4 bits
  - 64-QAM, 256-QAM widely used

# Error Detection

# Error Detection & Correction

- Error – bit pattern sent and bit pattern received are different. Possible causes
  - Framing errors
  - Noise...
- Error detection – detects error but may or may not correct
- Error correction – detects and corrects errors
  - Usually requires larger number of extra bits with data bits, not usually done in network protocols, we will skip



# Error Detection Techniques

- Basic Principle
  - Transmitter: For a given bit stream  $M$ , additional bits (called error-detecting bits or check bits) are calculated as a function of  $M$  and appended to the end of the data bits in  $M$
  - Receiver: On receiving the bit stream, separates the data bits, performs the same calculation on data bits, and compares the two results. A detected error occurs if there is a mismatch.
- Common Methods
  - Parity Check
  - Cyclic Redundancy Codes
  - Checksum

# Parity Check

- One extra “parity” bit is added to each word
  - Odd parity: bit added so as to make # of 1's odd
  - Even parity: makes total # of 1's even
- Detects any odd number of bit errors (error in 1bit, 3 bits, 5 bits ...), but can be fooled by any even number of errors
- Simple and easy to implement
- Not very robust against noise

# CRC

- Powerful error detection method, easily implemented in hardware
- Message (M) to be transmitted is appended with extra frame checksum bits (F), so that bit pattern transmitted (T) is perfectly divisible by a special “generator” pattern (P) - (divisor)
- At destination, divide received message by the same P. If remainder is nonzero, there is an error

- Let
  - $T = (k+n)$ -bit frame to be transmitted,  $n < k$
  - $M = k$ -bit message, the first  $k$  bits of  $T$
  - $F = n$ -bit FCS, the last  $n$  bits of  $T$
  - $P = n+1$  bits, generator pattern (predetermined divisor)
- The concept uses modulo-2 arithmetic
  - no carries/borrows;  $\text{add} \equiv \text{subtract} \equiv \text{XOR}$

- Extend M with n '0's to the right ( $\equiv 2^n M$ )(shift left by n bits)
- Divide extended message by P to get R ( $2^n M / P = Q + R/P$ )
- Add R to extended message to form T ( $T = 2^n M + R$ )
- Transmit T
- At receiver, divide T by P. Nonzero remainder  $\Rightarrow$  error

$$\frac{T}{P} = \frac{2^n M + R}{P} = Q + \frac{R}{P} + \frac{R}{P} = Q + \frac{R + R}{P} = Q$$

Note:  
Remainder  
R=F=FCS  
in these  
examples

Note:  $R+R=0$  in mod-2 arithmetic

$$0 \oplus 0 = 0$$

$$1 \oplus 1 = 0$$

- $M = 110011$ ,  $P = 11001$ ,  $R = 4$  bits
- Append 4 zeros to  $M$ , we get  $1100110000$

$$\begin{array}{r}
 - 2. \qquad \qquad \qquad 100001 \\
 11001 \overline{) 1100110000} \\
 \oplus 11001 \phantom{0000} \\
 \hline
 00001 \phantom{0000} \\
 \oplus 00000 \phantom{0000} \\
 \hline
 00010 \phantom{0000} \\
 \oplus 00000 \phantom{0000} \\
 \hline
 00100 \phantom{0000} \\
 \oplus 00000 \phantom{0000} \\
 \hline
 01000 \phantom{0000} \\
 \oplus 00000 \phantom{0000} \\
 \hline
 10000 \phantom{0000} \\
 \oplus 11001 \phantom{0000} \\
 \hline
 1001
 \end{array}$$

- 3.  $\therefore T = 110011$  1001

For each stage of division, if the number of dividend bits equals number of divisor  $P$  bits, then  $Q=1$ , otherwise  $Q=0$

- Can view CRC generation in terms of polynomial arithmetic also
- Any bit pattern  $\equiv$  polynomial in dummy variable  $X$  with the bits as coefficients as shown in the following example:
  - e.g.,  $M = 110011 \equiv 1 \cdot X^5 + 1 \cdot X^4 + 0 \cdot X^3 + 0 \cdot X^2 + 1 \cdot X + 1 \cdot X^0$   
 $\therefore M(X) = X^5 + X^4 + X + 1$

- CRC generation in terms of polynomial
  - Append n '0's  $\equiv X^n M(X)$
  - Modulo 2 division  $\rightarrow \frac{X^n M(X)}{P(X)} = Q(X) + \frac{R(X)}{P(X)}$
  - Transmit  $X^n M(X) + R(X) = T(X)$
  - At receiver:

$$\begin{aligned} \frac{X^n M(X) + R(X)}{P(X)} &= \frac{X^n M(X)}{P(X)} + \frac{R(X)}{P(X)} \\ &= Q(X) + \boxed{\frac{R(X) + R(X)}{P(X)}} \end{aligned}$$

→ If no error 0 always



- Commonly used polynomials,  $P(X)$ 
  - CRC-16 =  $X^{16} + X^{15} + X^2 + 1$
  - CRC-CCITT =  $X^{16} + X^{12} + X^5 + 1$
  - CRC-32 =  $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^{12} + X^4 + X^2 + X + 1$

- CRC can detect
  - All single-bit errors
  - All double-bit errors, as long as  $P(X)$  has a factor with at least three terms (as long as  $P$  has at least three 1s)
  - Any odd number of errors, as long as  $P(X)$  contains a factor  $(X+1)$
  - Any burst error for which the length of the burst is less than or equal to the length of the FCS
  - Many other larger burst errors
- Strong error detection scheme, easy implementation in hardware

# Checksum

- Another common error detection scheme
- Break up the bit streams into fixed length “words” (typically 16 bit)
- Transmitter
  - Do 1’s complement addition of all the words
  - Take the 1’s complement of the final sum and send it with the bit stream
- Receiver
  - Do one’s complement addition of all the words (including the checksum)
  - Should be all 1’s

# Example

- Input stream (in hex): 00 01 F2 03 F4 F5 F6 F7
- Stream sent : 00 01 F2 03 F4 F5 F6 F7 **22 0D**

Partial sum	$\begin{array}{r} 0001 \\ F203 \\ \hline F204 \end{array}$
Partial sum	$\begin{array}{r} F204 \\ F4F5 \\ \hline 1E6F9 \end{array}$
Carry	$\begin{array}{r} E6F9 \\ 1 \\ \hline E6FA \end{array}$
Partial sum	$\begin{array}{r} E6FA \\ F6F7 \\ \hline 1DDF1 \end{array}$
Carry	$\begin{array}{r} DDF1 \\ 1 \\ \hline DDF2 \end{array}$
Ones complement of the result	220D

(a) Checksum calculation by sender

Partial sum	$\begin{array}{r} 0001 \\ F203 \\ \hline F204 \end{array}$
Partial sum	$\begin{array}{r} F204 \\ F4F5 \\ \hline 1E6F9 \end{array}$
Carry	$\begin{array}{r} E6F9 \\ 1 \\ \hline E6FA \end{array}$
Partial sum	$\begin{array}{r} E6FA \\ F6F7 \\ \hline 1DDF1 \end{array}$
Carry	$\begin{array}{r} DDF1 \\ 1 \\ \hline DDF2 \end{array}$
Partial sum	$\begin{array}{r} DDF2 \\ 220D \\ \hline FFFF \end{array}$

(b) Checksum verification by receiver

- Better than parity check
  - Higher error detection capability
- Less effective than CRC
- Still used a lot in higher layer protocols because of simplicity and low overhead of computing the checksum

# Flow Control

# Flow Control

- What if sender sends the bit pattern too fast? Receiver may not be able to process the data as fast as the sender is sending it
  - Receiver can buffer, but buffer has finite size. After that is filled, data is 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

- Sender sends a frame
- Receiver receives frame & acknowledges it
- Sender waits to receive “ack” before sending next frame  
(If receiver is not ready to receive another frame it holds back the ack)
- One frame at a time is sent over the transmission line
- Simple, easy to implement



- Problems:
  - Works fine if there is only a few large frames to be sent
  - Usually we want smaller frames
    - Receiver's buffer size may be small
    - If a large frame is transmitted, the entire frame has to be retransmitted if there is an error
      - Longer the frame, larger the chance of an error
    - On a shared medium, do not want any one sender to occupy the medium for a long time (will see later)
  - With smaller frames, takes too long to send all frames

# Link Efficiency of Stop & Wait

- Bit Length of link in bits  $B$  = maximum possible number of bits that can be simultaneously present in the link
  - $B = R \times (D/V)$ 
    - $R$  = data rate of link in bps
    - $D$  = length of link in meters
    - $V$  = propagation speed in m/sec
- So for Stop & Wait, ideally we want frame length to be close to bit length of the link
  - The sender has enough bits to put out on the link while the earlier bits of the same frame reach the receiver
  - Link stays occupied for more time
- But we have also argued that we want frame sizes to be small
- Results in very poor line/link utilization for Stop & Wait, especially in fast links over longer distances
  - $B$  becomes very large, so impractical to have such large frames

- Consider a frame length of  $L$  bits
- Transmission delay = Time to put the frame into the link =  $L/R$
- Time for last bit of the frame to reach the receiver =  $(L/R) + (D/V)$
- Assume that processing delay at receiver is negligible
- Assume that transmission time of ACK is negligible (very small frame)
- Time for ACK to come to sender =  $(D/V)$
- Total time to transmit the frame  $T = (L/R) + 2(D/V)$
- Total no. of bits that could have been transmitted in this time =  $T \times R$
- Actual no. of bits transmitted =  $L$
- Line utilization =  $L/(T \times R) = L / (L + 2R(D/V))$

# Sliding Window Flow Control

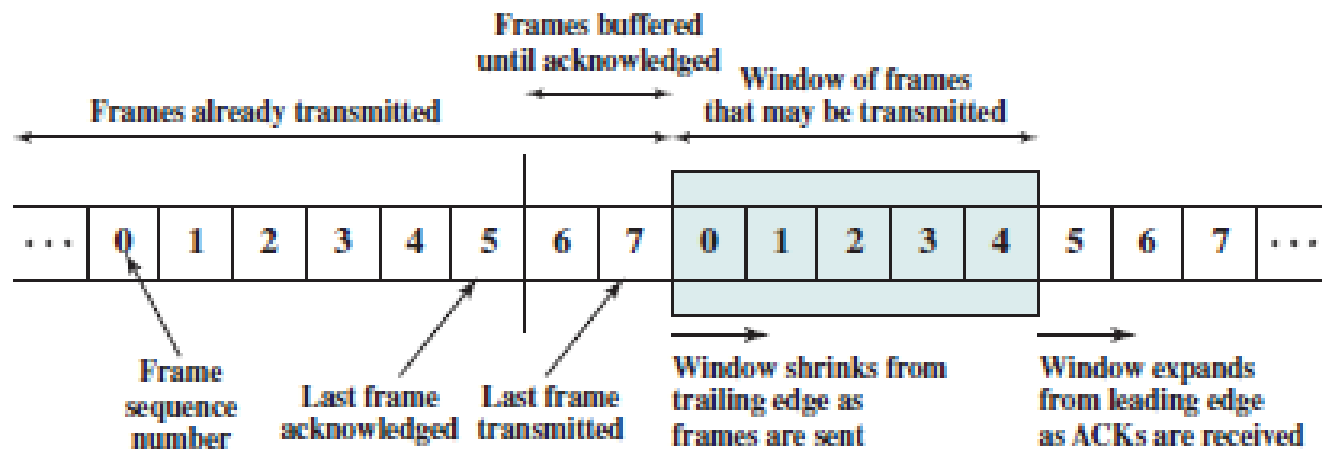
- Reduces line inefficiency problem of stop-and-wait by transmitting multiple frames without waiting for acknowledgement
- Suppose two stations A and B are connected by a full-duplex link



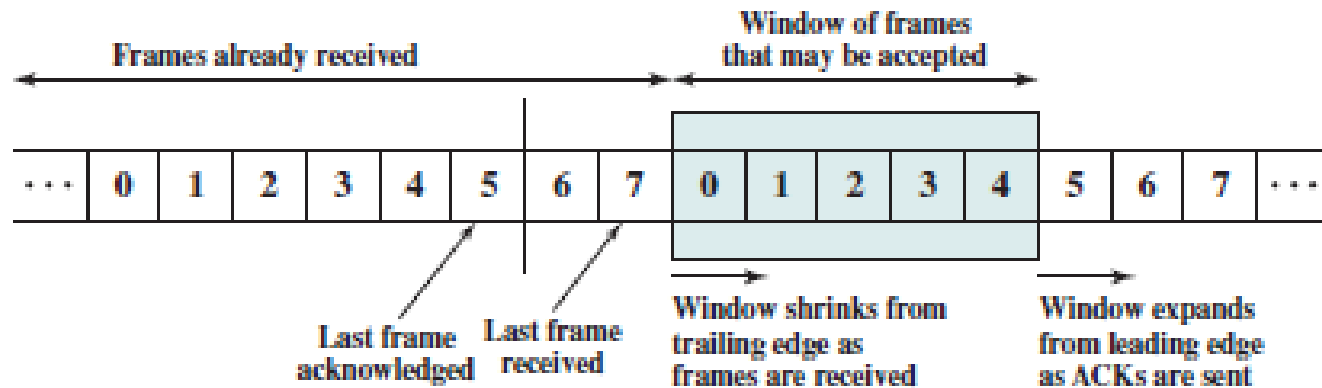
# Sliding Window (contd.)

- Station B allocates buffer space for  $W$  frames.
  - Thus B can accept  $W$  frames, and A is allowed to send  $W$  frames without waiting for any acknowledgement
- Sender maintains a list of sequence numbers that it is allowed to send (**sender window**)
- Receiver also maintains a list of sequence numbers that it is prepared to receive (**receiver window**)
- Since the sequence number to be used occupies a field in the frame, it is clearly of bounded size
  - For a  $k$ -bit field, the range of sequence numbers is 0 through  $2^k - 1$ , and frames are numbered modulo  $2^k$
  - Maximum window size =  $2^k - 1$  (why?)

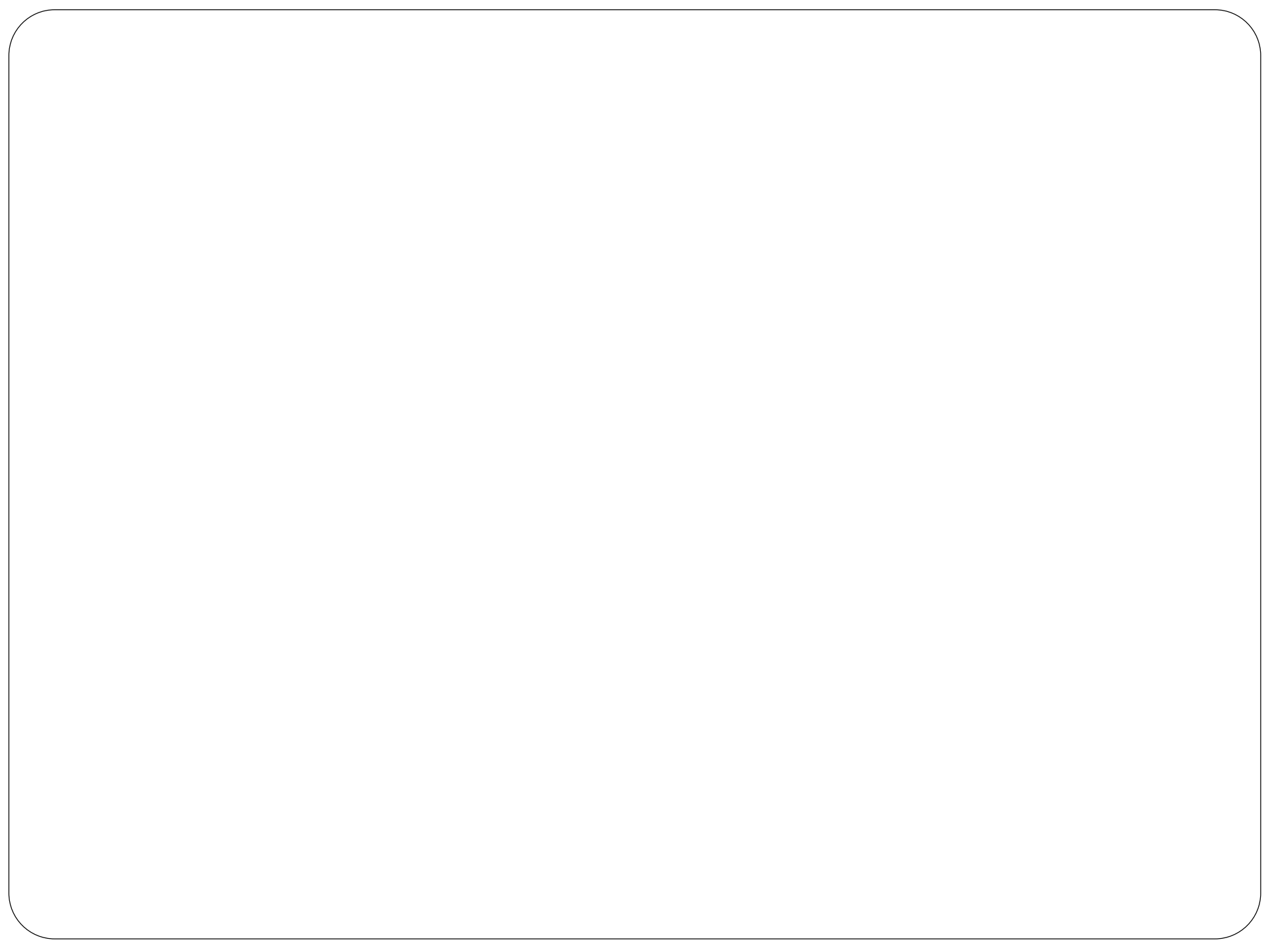
- Each frame is labeled with a sequence number
  - To keep track of the frames which have been acknowledged
  - B acknowledges a frame by sending an ACK/RR that includes the sequence number of the next frame expected. This also explicitly announces that B is prepared to receive the next  $W$  frames, beginning with the number specified
- This scheme can be used to acknowledge multiple frames
  - B could receive frames 2,3,4 but withhold ACK until frame 4 has arrived. By returning an ACK with sequence number 5, B acknowledges frames 2,3,4 at one time



(a) Sender's perspective



(b) Receiver's perspective





# Sliding Window (contd.)

- The actual window size need not be the maximum possible size for a given sequence number length
  - For a 3-bit sequence number, a window size of 5 can also be configured
- If two stations exchange data, each need to maintain two windows. To save communication capacity, a technique called **piggybacking** is used
  - Each data frame includes a field that holds the sequence number of that frame plus a field that holds the sequence number used for ACK
  - If a station has an ACK but no data to send, it sends a separate ACK frame

# Error Control

# Error Control

- Ensures finally received bit pattern is same as sent bit pattern, though may require more than one transmission
- Type of frame loss
  - Lost frames (How can it be lost completely??)
  - Damaged Frames (error detected)
- Two types of error control
  - Forward error control:
    - Error recovery by correction at the receiver [Forward Error Correction (FEC)]
    - Requires extra error correcting bits to be added to data bits (Ex. Hamming code)
    - No. of bits needed large for even small number of bit errors
  - Backward error control:
    - Error recovery by retransmission [Automatic Repeat Request (ARQ)]

# ARQ

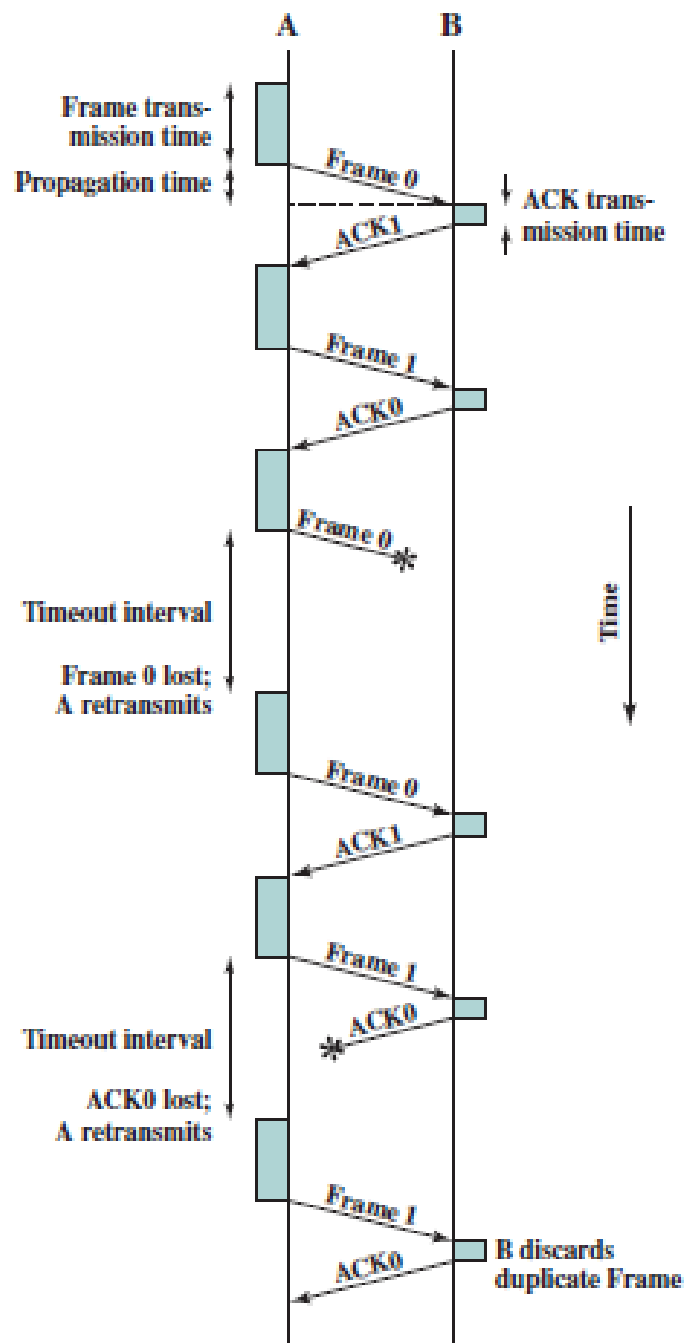
- Basic idea
  - Break up bit pattern into fixed length frames
  - Add error detection bits with each frame
  - Ask transmitter to retransmit frames that show error or that are expected but not received within a timeout
  - Reconstruct final received bit pattern when all frames received successfully

# Stop & Wait ARQ

- Sender transmits message frame
- Receiver checks received frame for errors, sends ACK
- Sender waits for ACK to send next frame
- Two possibilities:
  - Receiver does not receive the frame or receives a damaged frame
    - Receiver simply discards a damaged frame
    - Sender will timeout, resend the frame
    - This continues until an ACK is received

# Stop & Wait ARQ (contd.)

- Receiver receives the frame but the ACK is lost or damaged
- Sender times out, or discards the ACK, sends the frame again
- So receiver may receive the frame more than once
- Handling duplicate frames
  - Add a 1-bit frame number, 0 or 1, to each frame
  - ACKS includes a frame number – ACK0 and ACK1
    - ACK0 implies sender has received frame 1 successfully and vice-versa
  - Only need a 1-bit frame number alternating 1 and 0 since they are sent one at a time
    - Two frames are never together on the line



- Pros

- Simple to implement
- Only 1-bit frame number needed

- Con

- Extremely inefficient (as already seen in flow control)
  - Only one frame can be sent at a time
  - Sender has to wait for an ack even if it has more frames to send and even if the frame is received successfully



# Go-back-N ARQ

- The sender can send a sequence of frames without waiting for others
- The frames are numbered sequentially
- Window size – the maximum number of unacknowledged frames that can be there
- Behavior on no error – similar to sliding window
- If the receiver detects an error on a frame, it sends a NACK/REJ for that frame. The receiver will discard all future frames received until the frame in error is correctly received
- The sender, when it receives a NACK/REJ or times out, must retransmit the frame in error plus all succeeding frames. (Sender must maintain a copy of each unacknowledged frame)

- Maximum window size with k-bit sequence number is  $2^k - 1$
- What if not?
  - Suppose window size is 8 ( $= 2^k$ ) for  $k = 3$
  - Sender sends frame 0, receives ACK/RR 1
  - Sender sends frames 1, 2, 3, 4, 5, 6, 7, 0 (next 8 frames)
  - Sender gets another ACK/RR1
  - Ambiguity:
    - Did all 8 frames get through successful?
    - Did all 8 frames were lost/damaged?

# Selective-Reject ARQ

- The only frames retransmitted are those that receive a NACK or which timeout
- Pro: Can save retransmissions by sending only frames that are in error
- Con:
  - Requires more buffer space, as receiver must keep enough buffer space to save frames received successfully after the frame in error
  - More complicated logic to reinsert the lost frame in the sequence when received successfully later
  - Lower window size of  $2^{k-1}$  (why?)

# Summary

- We now know how to make only two machines communicate connected directly with a link
  - Encode data to be send with signals
  - Break data into frames
  - Make sure sender and receiver are synchronized for proper frame receive
  - Detect errors in frames (error detection) as well as lost frames (timeout)
  - Use error control techniques to ensure all frames (and therefore all data) are received at the receiver correctly eventually
  - Ensure fast sender cannot overwhelm a slow receiver
- Next step: More than one machine sharing a direct link