

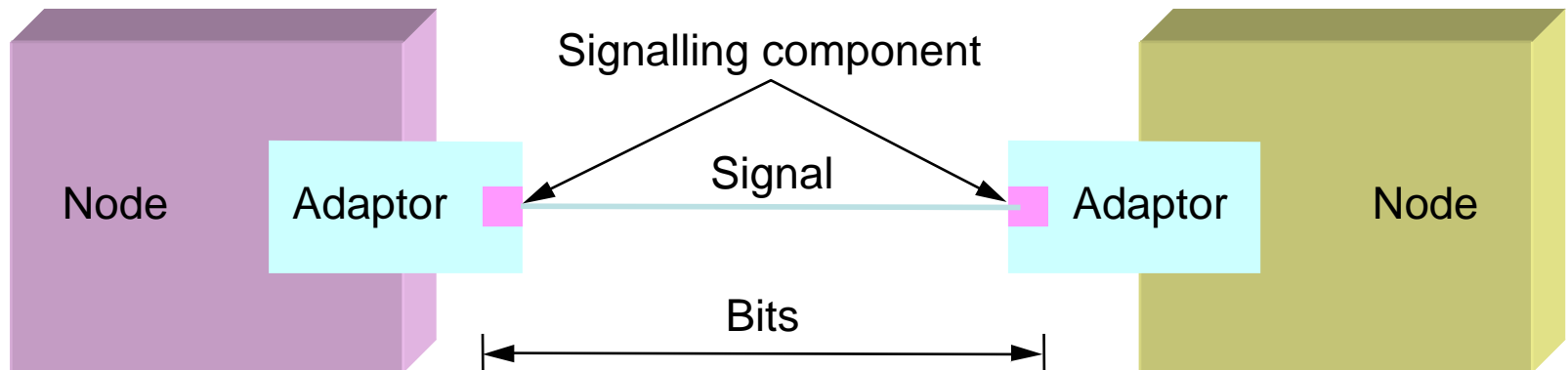
Encoding

Point-to-Point Links

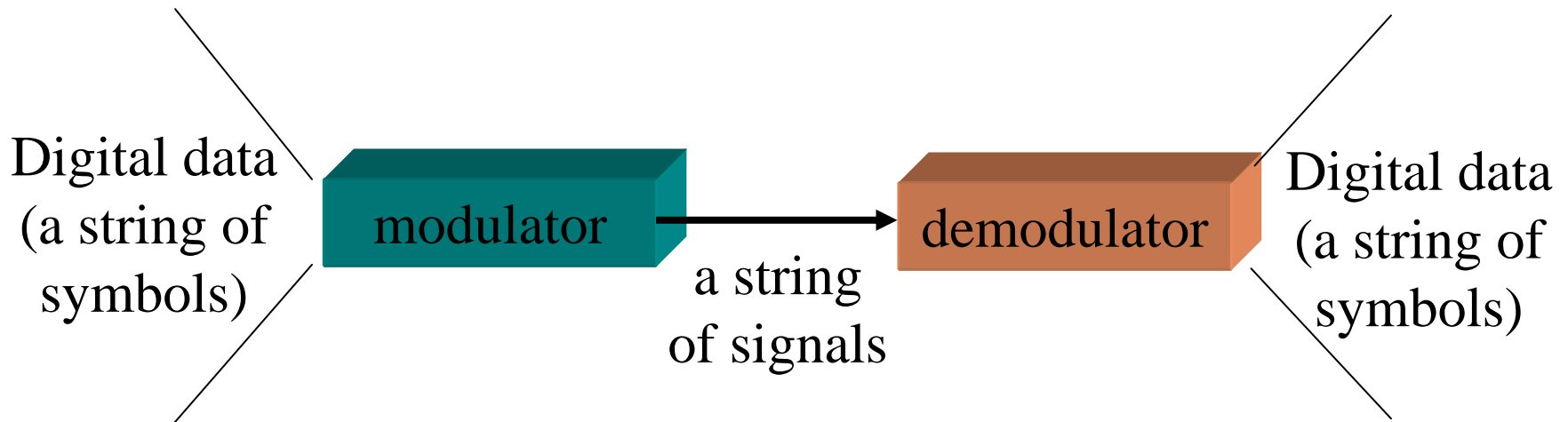
- Reading: Peterson and Davie, Ch. 2
- Hardware building blocks
- **Encoding**
- Framing
- Error Detection
- Reliable transmission
 - Sliding Window Algorithm

Encoding

- Signals propagate over a physical medium
 - modulate electromagnetic waves
 - e.g., vary voltage
- Encode **binary data onto signals** that propagate



Encoding



- Problems with signal transmission
 - **Attenuation**: signal power absorbed by medium
 - **Dispersion**: a discrete signal spreads in space
 - **Noise**: random background “signals”

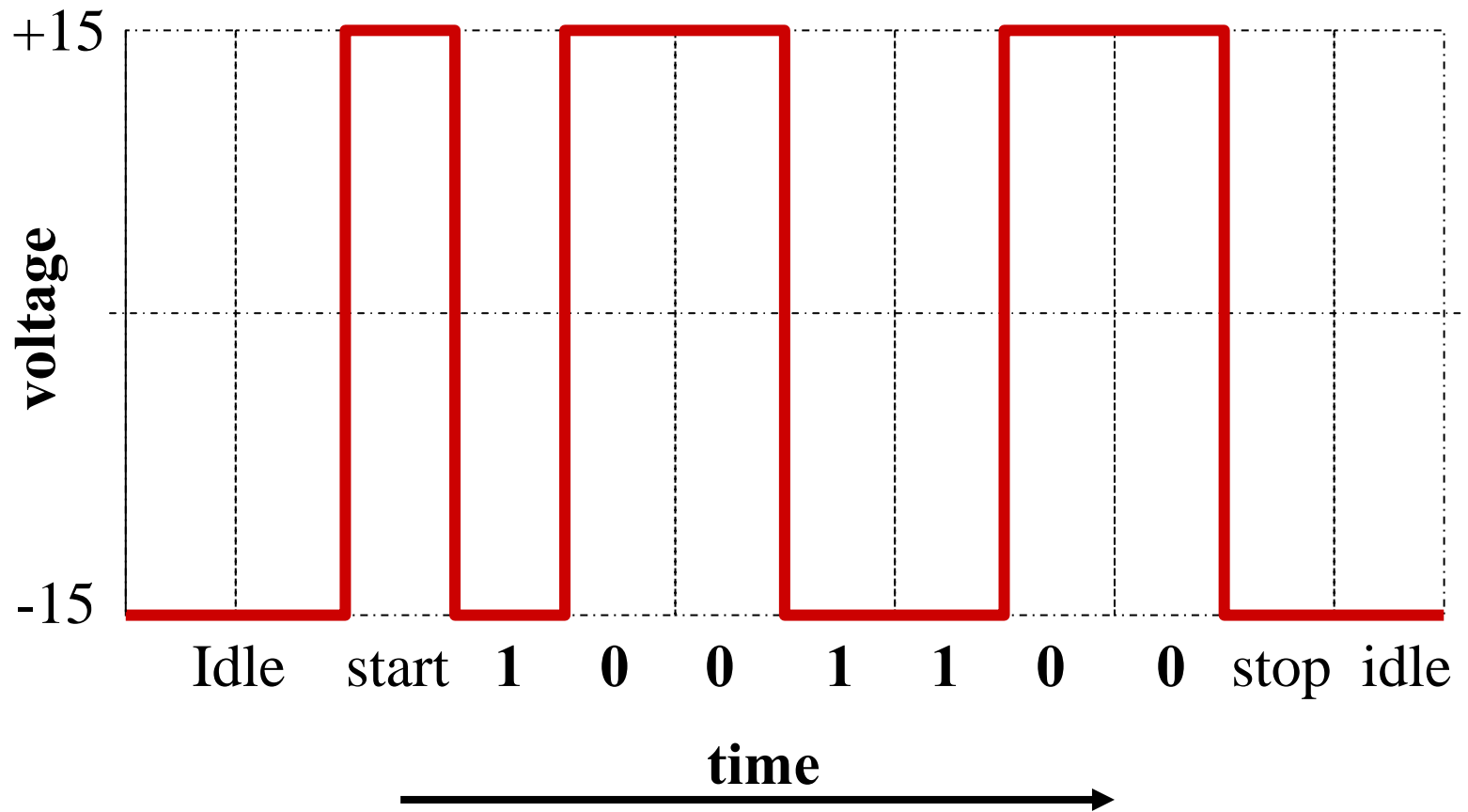
Advantages of Digital Transmission over Analog

- Reasonably low-error rates over arbitrary distances
 - Calculate/measure effects of transmission problems
 - Periodically interpret and regenerate signal
- Simpler for multiplexing distinct data types (audio, video, e-mail, etc.)
- Examples of modulators-demodulators (modems)
 - Electronic Industries Association (EIA) standard RS-232(-C)
 - International Telecommunications Union (ITU) standard V.32, 9.6 kbps modem

RS-232(-C)

- Communication between computer and modem
- Uses two voltage levels (+15V, -15V), a **binary voltage** encoding
- Data rate limited to **19.2 kbps** (RS-232-C); raised in later standards
- Characteristics
 - **Serial**: one signaling wire, one bit at a time
 - **Asynchronous**: line can be idle, clock generated from data
 - **Character-based**: send data in 7- or 8-bit characters

RS-232 Timing Diagram



RS-232

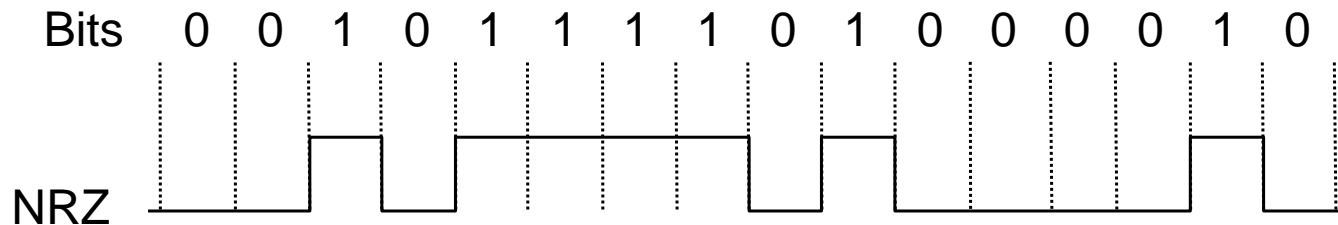
- One bit per clock
- Voltage **never returns to 0V** (0V is a dead / disconnected line)
- -15V is both idle and “1”; initiates the send by pushing to 15V for one clock (start bit)
- Minimum delay between character transmissions idle for one clock at -15V (stop bit)
- One character leads to **2+ voltage transitions**
- Total of 9 bits for 7 bits of data (**78% efficient**)
- Start and stop bits also provide **framing**

Binary Voltage Encoding

- NRZ (non-return to zero)
- NRZI (NRZ inverted)
- Manchester (used by IEEE 802.3, 10 Mbps Ethernet)
- 4B/5B (8B/10B) in Fast Ethernet

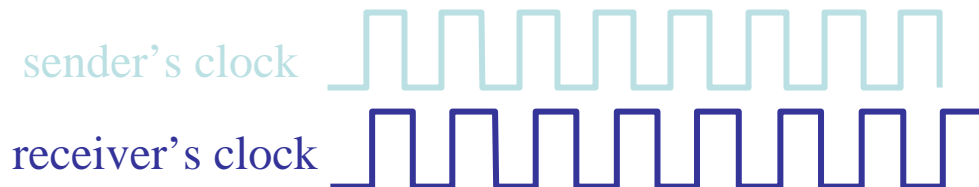
Non-Return to Zero (NRZ)

- Encode binary data onto signals
 - e.g., **0 as low** signal and **1 as high** signal
 - voltage does not return to zero between bits
 - known as Non-Return to Zero (NRZ)



Problem: Consecutive 1s or 0s

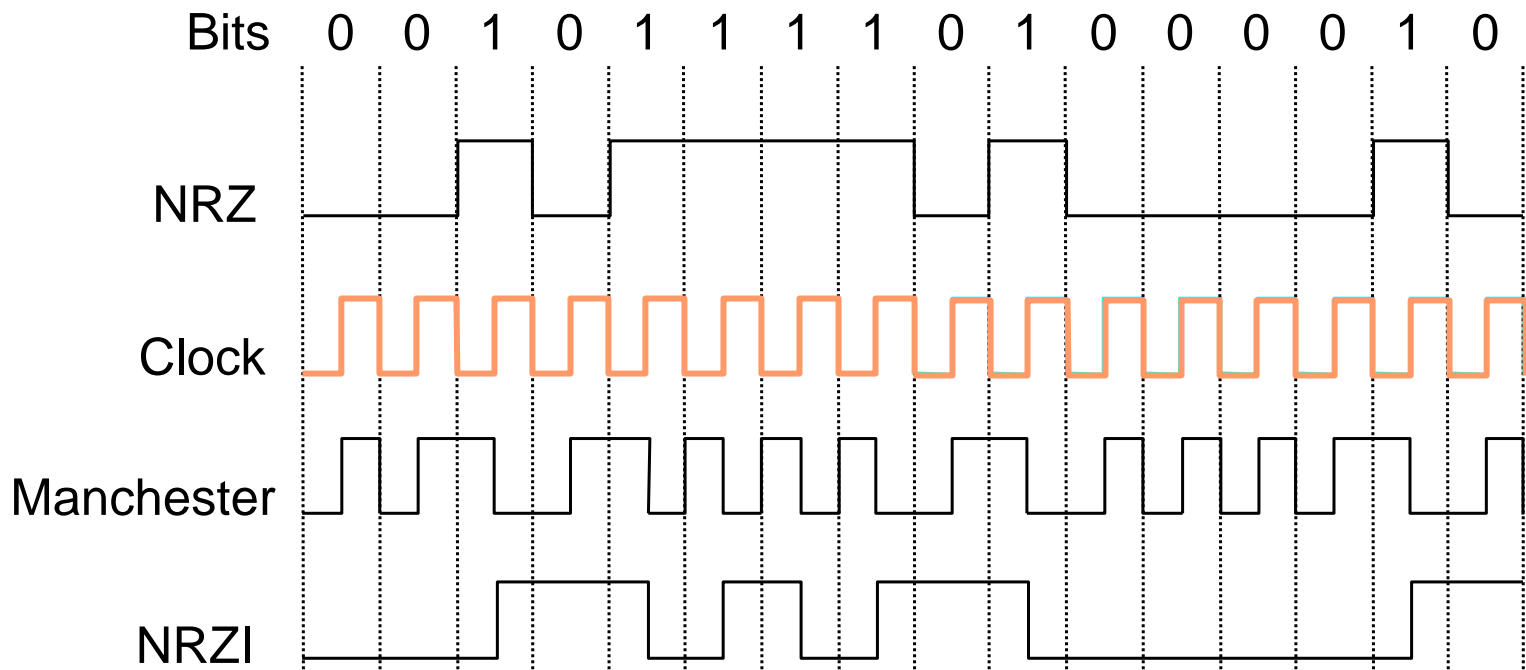
- Low signal (0) may be interpreted as **no signal**
- High signal (1) leads to **baseline wander**
- Unable to **recover clock**
 - sender's and receiver's clock have to be precisely synchronized
 - receiver resynchronizes on each signal transition
 - **clock drift** in long periods without transition



Alternative Encodings

- Non-Return to Zero Inverted (NRZI)
 - make a **transition from current signal** (switch voltage level) to encode/transmit a “one”
 - **stay at current signal** (maintain voltage level) to encode/ transmit a “zero”
 - solves the problem of consecutive ones (shifts to 0s)
- Manchester (in IEEE 802.3 – 10 Mbps Ethernet)
 - split cycle into two parts
 - send **high--low** for “1”, **low--high** for “0”
 - transmit XOR of NRZ encoded data and the clock
 - only 50% efficient (1/2 bit per transition)

Different Encoding Schemes



4B/5B Encoding

- Every 4 consecutive bits of data encoded in a 5-bit code (symbol)
 - 4-bit pattern is “translated” to a 5-bit pattern (not addition)
- 5-bit codes selected to have no more than one leading 0 and no more than two trailing 0s
 - 00xxx (8 symbols) and xx000 (4 symbols) are illegal
 - 5 free symbols (**non-data**)
- Thus, never gets more than three consecutive 0s
- Resulting 5-bit codes are transmitted using NRZI
- Achieves 80% efficiency

Binary Voltage Encoding

- Problem: wide frequency range required, implying
 - Significant dispersion
 - Uneven attenuation
- Prefer to use **narrow frequency band** (carrier frequency)
- Types of modulation
 - Amplitude (AM)
 - Frequency (FM)
 - Phase / phase shift
 - Combination of these (e.g. QAM)

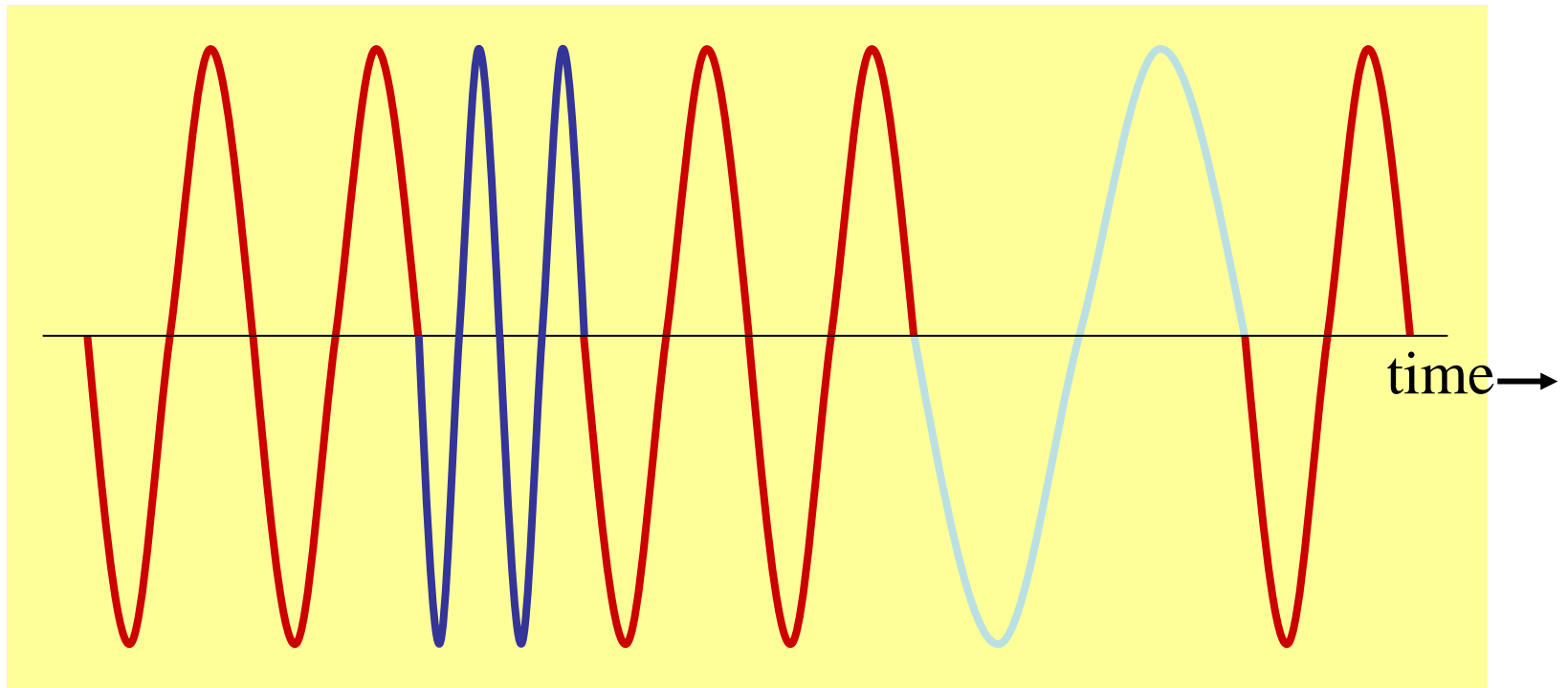
Amplitude Modulation

idle idle 1 idle idle 0 idle idle



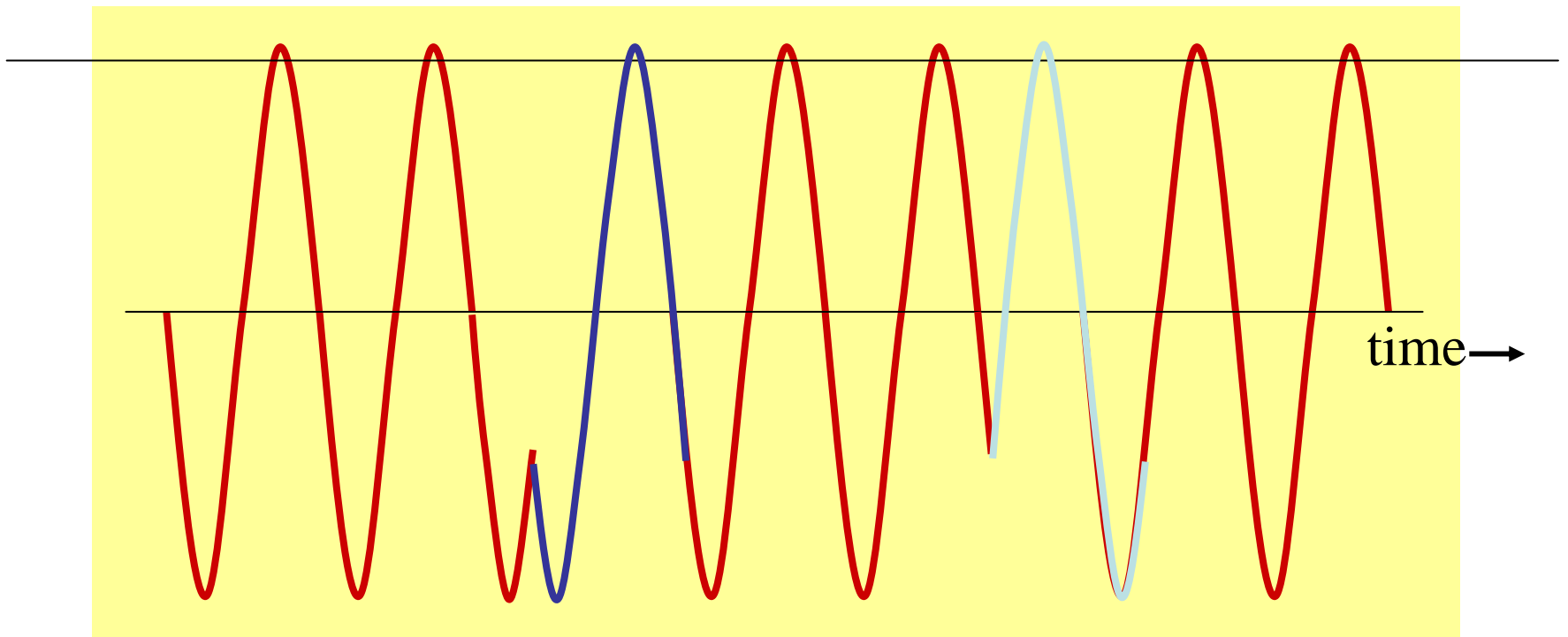
Frequency Modulation

idle idle 1 idle idle 0 idle

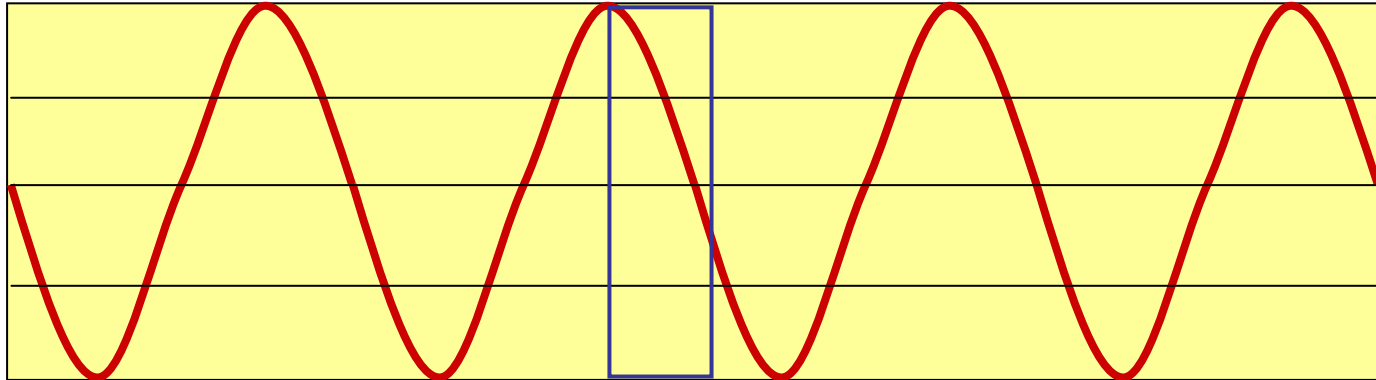


Phase Modulation

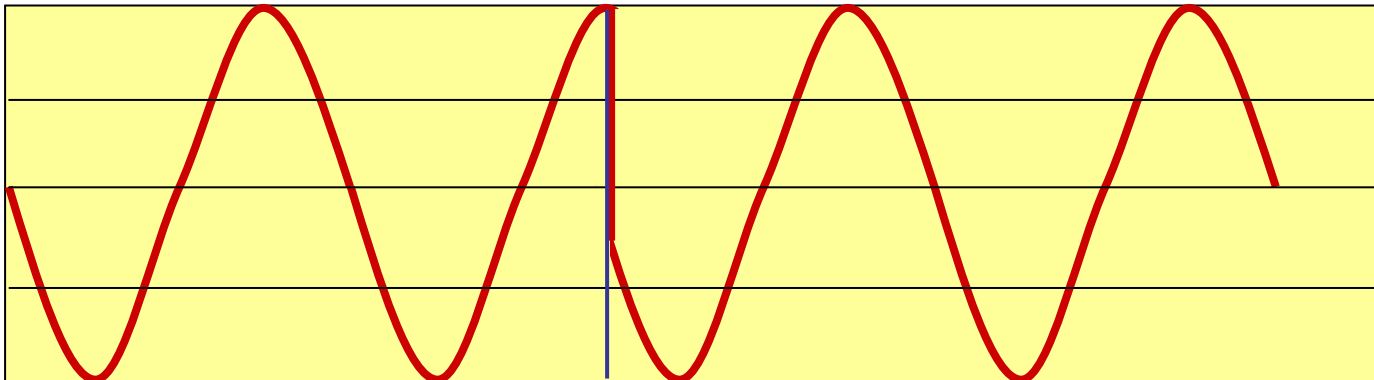
idle idle 1 idle idle 0 idle idle



Phase Shift in Carrier Frequency



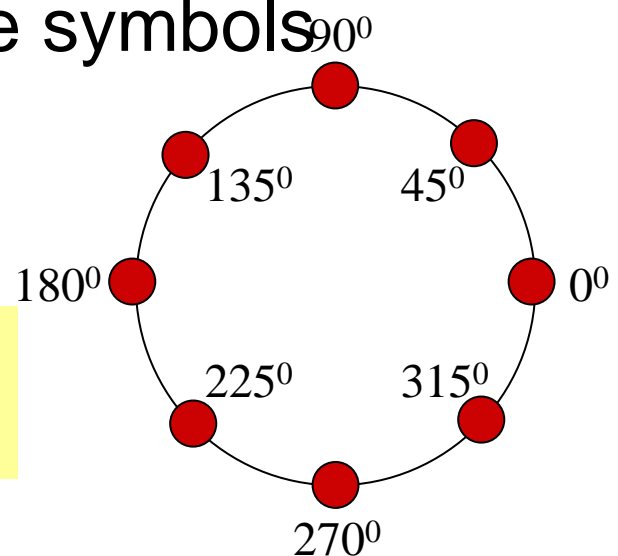
→ | ← 108 degrees difference in phase
→ | ← collapse for 108 degrees shift



Phase Modulation Algorithm

- Send carrier frequency for one period
- Perform phase shift
- Shift value encodes symbol
 - value in range $[0, 360^\circ]$
 - multiple values for multiple symbols
 - represent as circle

8-symbol
example



ITU's V.32 - 9.6 kbps

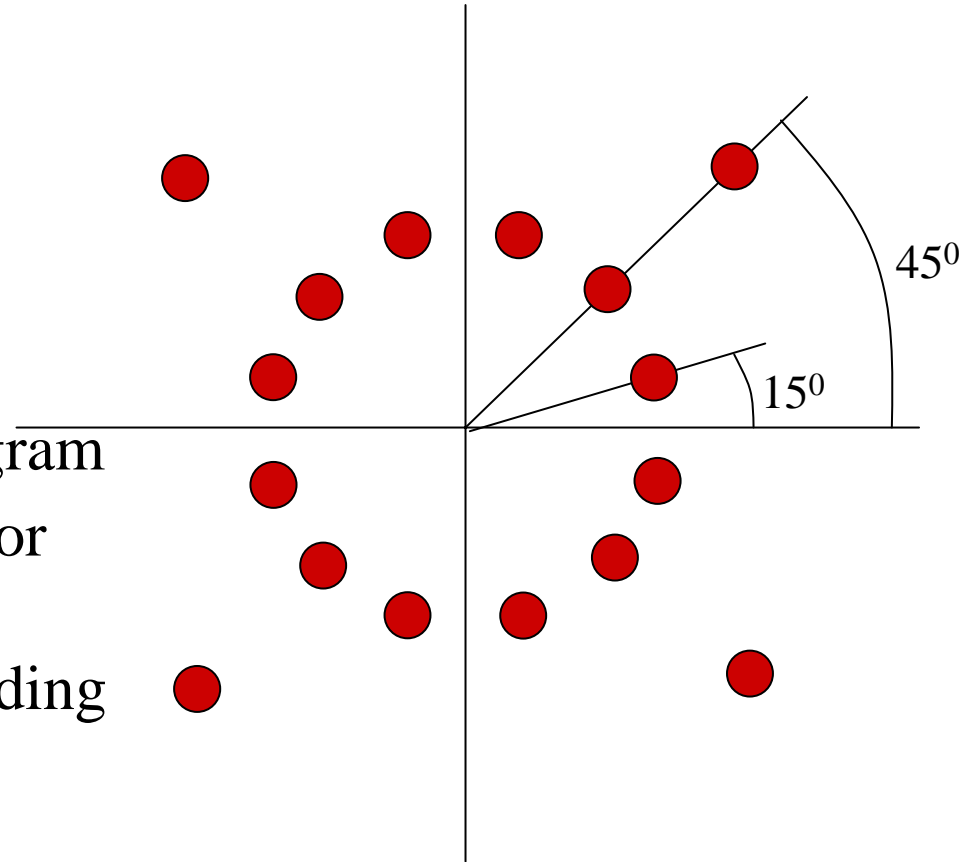
- Communication between modems
- Analog phone line
- Uses a combination of amplitude and phase modulation
 - known as Quadrature Amplitude Modulation (QAM)
- Sends one of 16 signals each clock cycle
 - transmits at 2400 baud, *i.e.*, 2,400 symbols per second

Constellation Pattern for V.32 QAM

For a given symbol:

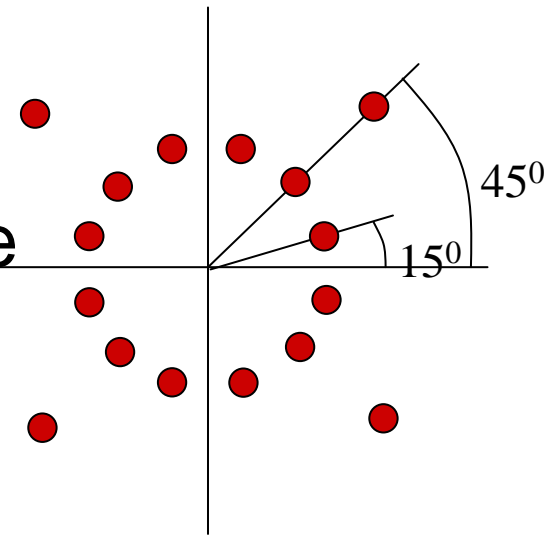
1. perform phase shift
2. change to new amplitude

- Points in constellation diagram
 - chosen to maximize error detection
 - process called trellis coding



Quadrature Amplitude Modulation

- Same algorithm as phase modulation
- Can also change signal amplitude
- 2-dimensional representation
 - angle is phase shift
 - radial distance is new amplitude
- Each symbol contains $\log_2 16 = 4$ bits
 - data rate is thus $4 \times 2400 = 9600$ bps



16-symbol
example
(V.32)

Generalizing the Examples

- What limits baud rate?
- What data rate can a channel sustain?
- How is data rate related to bandwidth?
- How does noise affect these bounds?
- What else can limit maximum data rate?

Bit Rate and Baud Rate

- Bit rate is **bits** per second
- Baud rate is “**symbols**” per second
- If each symbol contains 4 bits then data rate is 4 times the baud rate

What Limits Baud Rate ?

- Baud rates are typically limited by **electrical signaling properties**
- No matter how small the voltage or how short the wire, **changing voltages takes time**
- Electronics are **slow** as compared to optics

What data rate can a channel sustain ?
How is data rate related to bandwidth ?

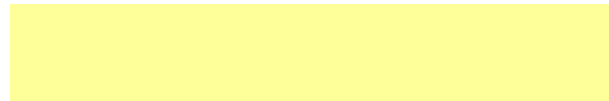
- Transmitting N distinct signals over a noiseless channel with bandwidth B , max. data rate can be $2B \log_2 N$
- This observation is a form of **Nyquist's Sampling Theorem**
 - We can reconstruct any waveform with no frequency component above some frequency “ F ” using only samples taken at frequency $2F$

What else (besides noise) can limit maximum data rate ?

- Transitions between symbols introduce high frequency components into the transmitted signal
- Such components cannot be recovered (by Nyquist's Theorem), and some information is lost
- Examples:
 - Pulse modulation uses only a single frequency (with different phases) for each symbol, but the transitions can require very high frequencies
 - Binary voltage encodings (0 Hz within symbols)
 - Eye diagrams show voltage traces for all transitions

How does Noise Affect these Bounds ?

- In-band (not high-frequency) noise blurs the symbols, reducing the number of symbols that can be reliably distinguished
- Shannon extended Nyquist's work to channels with additive white Gaussian noise (a good model for thermal noise)
- From **Shannon's Theorem** :
Max. channel capacity $C = B \log_2 (1+S/N)$



Summary of Encoding

- Problems: attenuation, dispersion, noise
- Digital transmission allows periodic **regeneration**
- Variety of binary voltage encodings
 - High frequency components limit to short range
 - More voltage levels provide higher data rate
- Carrier frequency and modulation
 - Amplitude, frequency, phase, and combination (QAM)
- Nyquist (noiseless) and Shannon (noisy) limits on data rates

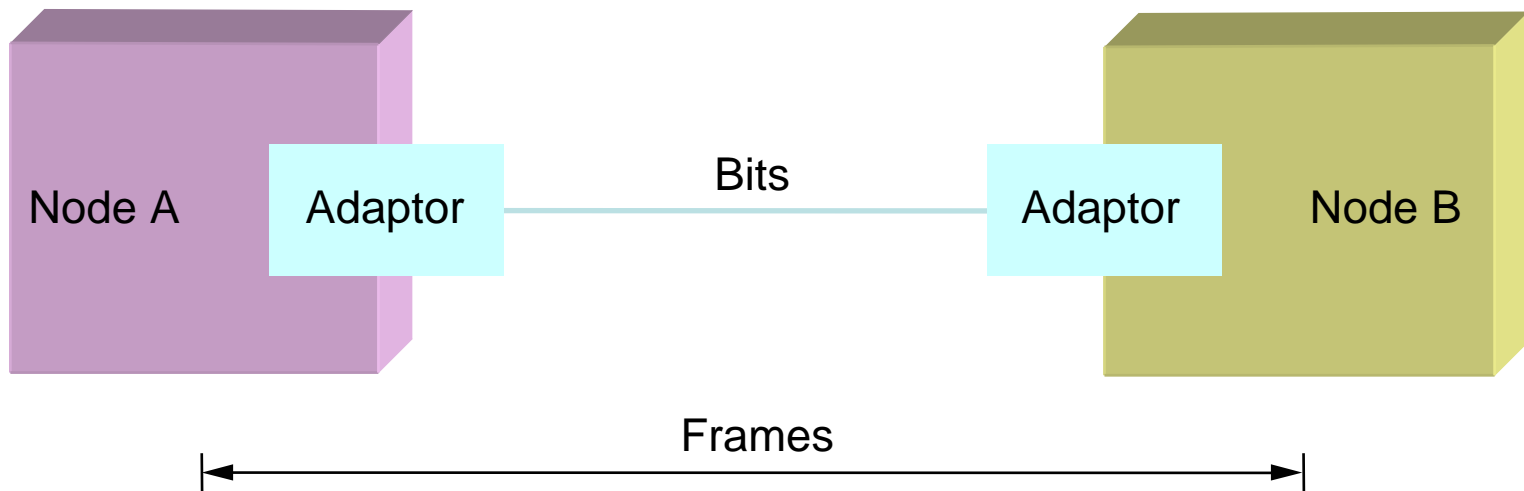
Framing

Point-to-Point Links

- Reading: Peterson and Davie, Ch. 2
- Hardware building blocks
- Encoding
- **Framing**
- Error Detection
- Reliable transmission
 - Sliding Window Algorithm

Framing

- Breaks continuous stream/sequence of bits into a frame and **demarkates units of transfer**
- Typically implemented by network adaptor
 - Adaptor fetches/deposits frames out of/into host memory



Advantages of Framing

- **Synchronization** recovery
 - consider continuous stream of unframed bytes
 - recall RS-232 start and stop bits
- **Multiplexing** of link
 - multiple hosts on shared medium
 - simplifies multiplexing of logical channels
- Efficient **error detection**
 - frame serves as unit of detection (valid or invalid)
 - error detection overhead scales as $\log N$

Problem ... ?

Recognizing exactly the **boundaries** of a frame

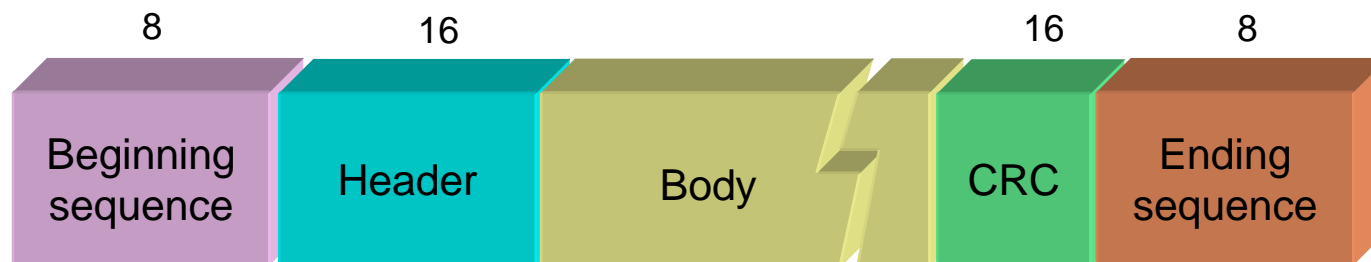
Must determine the **first and last bit** of a frame

Approaches

- Organized by **end of frame detection** method
- Approaches to framing
 - sentinel (marker, like C strings)
 - length-based (like Pascal strings)
 - clock-based
- Other aspects of a particular approach
 - bit- or byte-oriented
 - fixed- or variable-length
 - data-dependent or data-independent length

Framing with Sentinels

- End of frame: special byte or bit **pattern**
- Choice of end of frame marker
 - valid data byte or bit sequence e.g. 01111110
 - physical signal not used by valid data symbol



Sentinel Based Approach

- Problem: special pattern **appears in the payload**
- Solution: ***bit stuffing***
 - sender: insert 0 after five consecutive 1s
 - receiver: delete 0 that follows five consecutive 1s



Sentinel Based Approach

- Problem: **equal size frames** are not possible
 - frame length is data-dependent
- Sentinel based framing examples
 - High-Level Data Link Control (HDLC) protocol
 - Point-to-Point Protocol (PPP)
 - ARPANET IMP-IMP protocol
 - IEEE 802.4 (token bus)

Sentinels: HDLC

- Developed by IBM, standardized by OSI
- **Bit-oriented, variable-length, data-dependent**
- Special bit pattern 01111110 marks end of frame
- Insert 0 after pattern 011111 in data (bit stuffing)
- At receiver end, if the frame received is:
 - 01111110
 - bit stuffed, therefore receive only 011111
 - error in end of frame marker, lose two frames
 - 011111110: end of frame
 - 011111111: error, lose one or two frames

Sentinels: PPP

- **Byte-oriented, variable-length, data-dependent**
- Special flag 0111110 for *start-of-text*
 - address and control field uses default values (FF / 8E)
 - protocol field used for demultiplexing (IP,LCP,...)
- LCP (Link Control Protocol) send control messages
 - establishes link between two peers
 - negotiates payload and checksum size
- Insert 0 after pattern 011111 in data (bit stuffing)



Sentinels: ARPANET IMP-IMP

- Interface Message Processors (IMP's): packet-switching nodes in the original ARPANET
- **Byte-oriented, variable-length, data-dependent**
- Special bytes (aid to understand frame format below)
 - DLE: data link escape
 - STX/ETX: start and end of transmission
- DLE byte in data sent as two DLE's (byte stuffing)

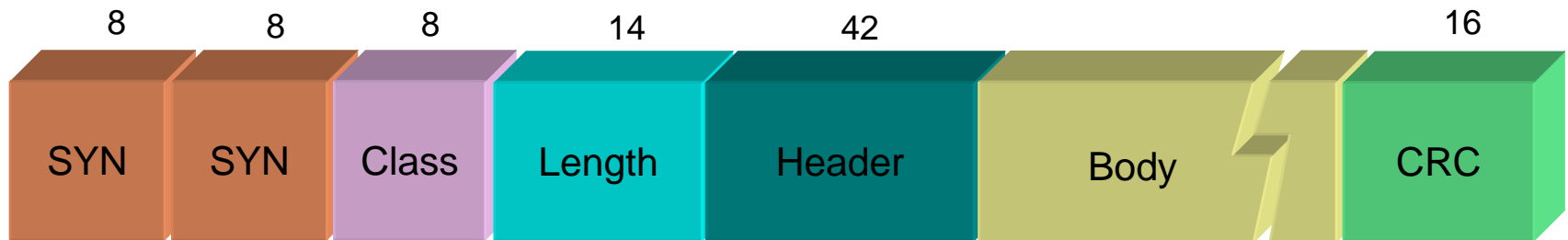


Sentinels: IEEE 802.4 (token bus)

- Alternative to Ethernet (802.3) with fairer arbitration
- End of frame marked by **encoding violation**, *i.e.*, physical signal not used by valid data symbol
- Recall Manchester encoding
 - low-high means “0”, high-low means “1”, low-low and high-high are invalid
- **Byte-oriented, variable-length, data-independent**
- Technique also applicable to bit-oriented framing
- Another example: Fiber Distributed Data Interface (FDDI) uses 4B/5B

Length-based Framing

- Include payload **length in header**
- e.g., DDCMP (byte-oriented, variable-length)
- e.g. RS-232 (bit-oriented, implicit fixed length)

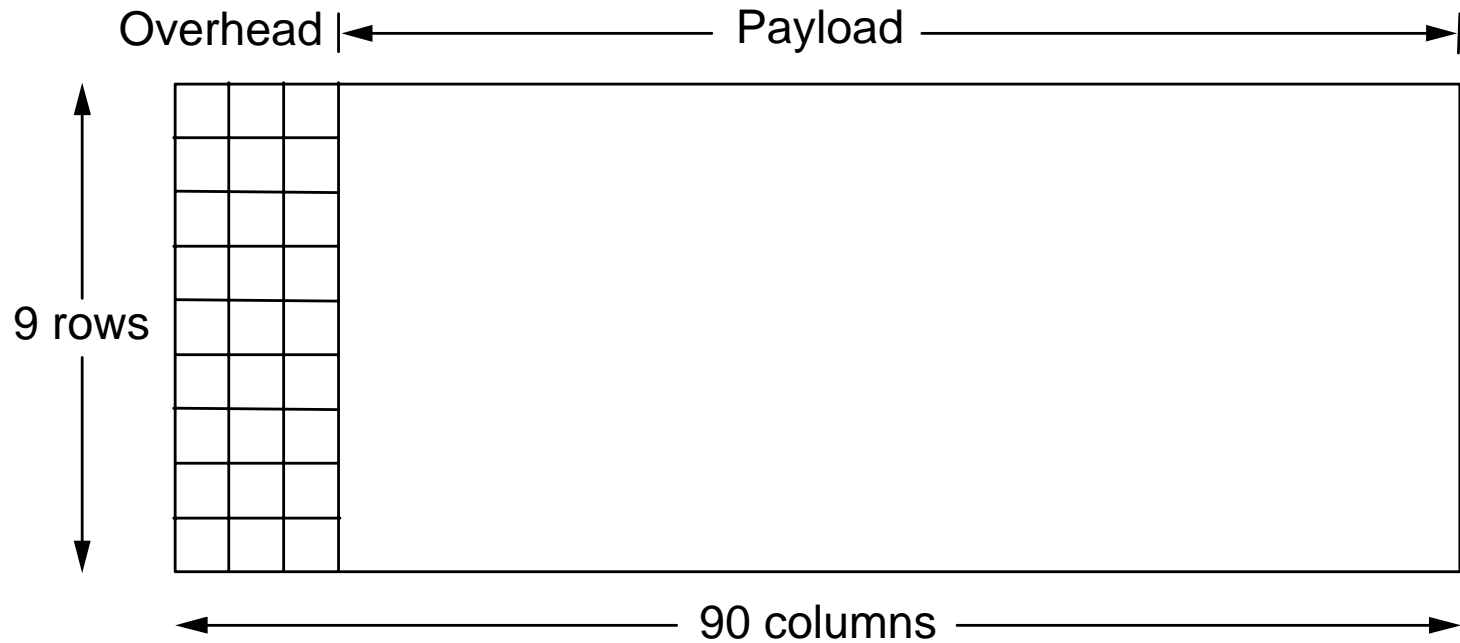


- Problem: count field corrupted
- Solution: catch when CRC fails

Clock-based Framing

- Continuous stream of **fixed-length frames**
 - each frame is 125us long (all STS formats) (why?)
- Clocks must remain **synchronized**
- e.g., SONET: Synchronous Optical Network
 - dominated standard for long distance transmission
 - multiplexing of low-speed links onto one high-speed link
 - byte-interleaved multiplexing
 - payload bytes are scrambled (data XOR 127 bit-pattern)
 - STS- n (STS-1 = 51.84 Mbps)

SONET Frame Format (STS-1)

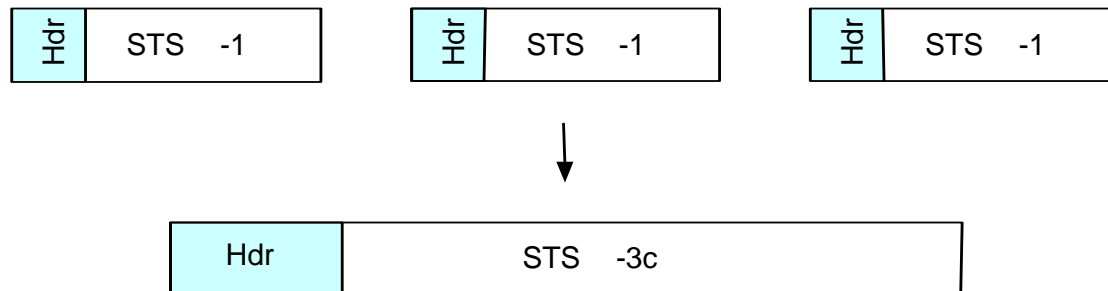


Clock-based Framing

- Problem: how to **recover** frame synchronization
 - 2-byte synchronization pattern starts each frame (unlikely to occur in data)
 - wait until pattern appears in same place repeatedly
- Problem: how to **maintain** clock synchronization
 - NRZ encoding, data scrambled (XOR'd) with 127-bit pattern
 - creates transitions
 - also reduces chance of finding false sync. pattern

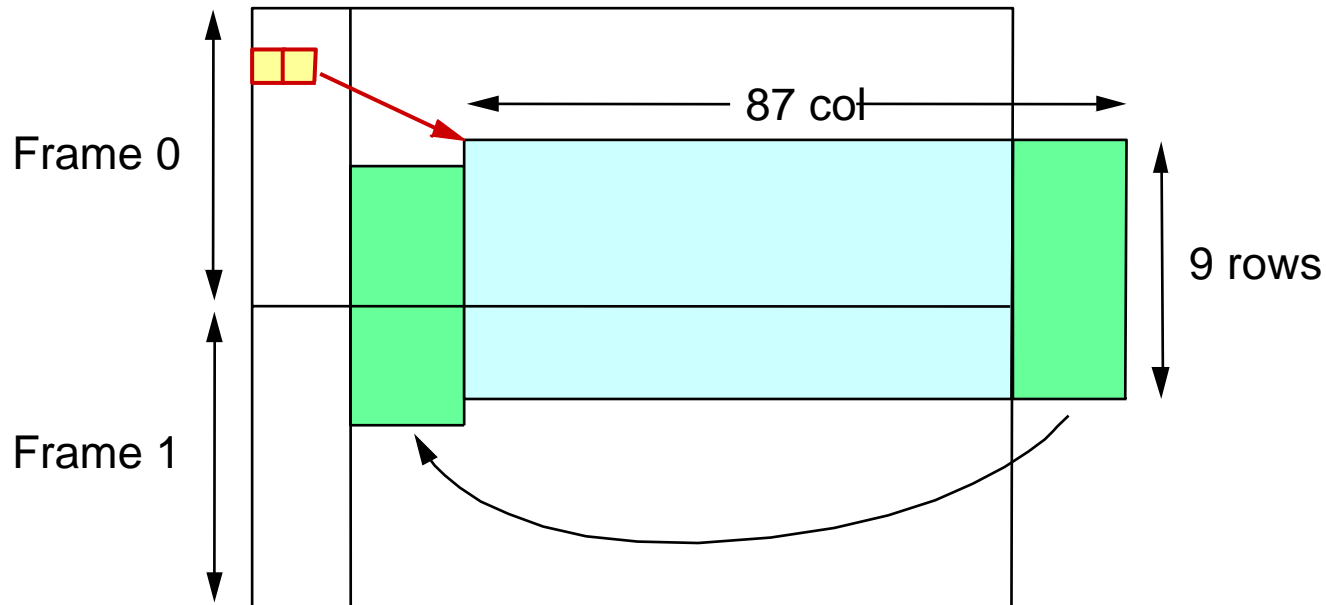
SONET Frame Merging

- STS-1 **merged bitwise round-robin** into STS-3
 - unmerged (single-source) format called STS-3c
- Problem: simultaneous synchronization of many distributed clocks
 - not too difficult to synchronize clocks such that first byte of all incoming flows arrives just before sending first 3 bytes of outgoing flow (**buffering ? delays ?**)



Clock-based Framing

- Problem: simultaneous synchronization of many distributed clocks
- Solution: payload frame **floats** within clock frame, part of overhead specifies first byte of payload



Error Detection

Point-to-Point Links

- Reading: Peterson and Davie, Ch. 2
- Hardware building blocks
- Encoding
- Framing
- **Error Detection**
- Reliable transmission
 - Sliding Window Algorithm

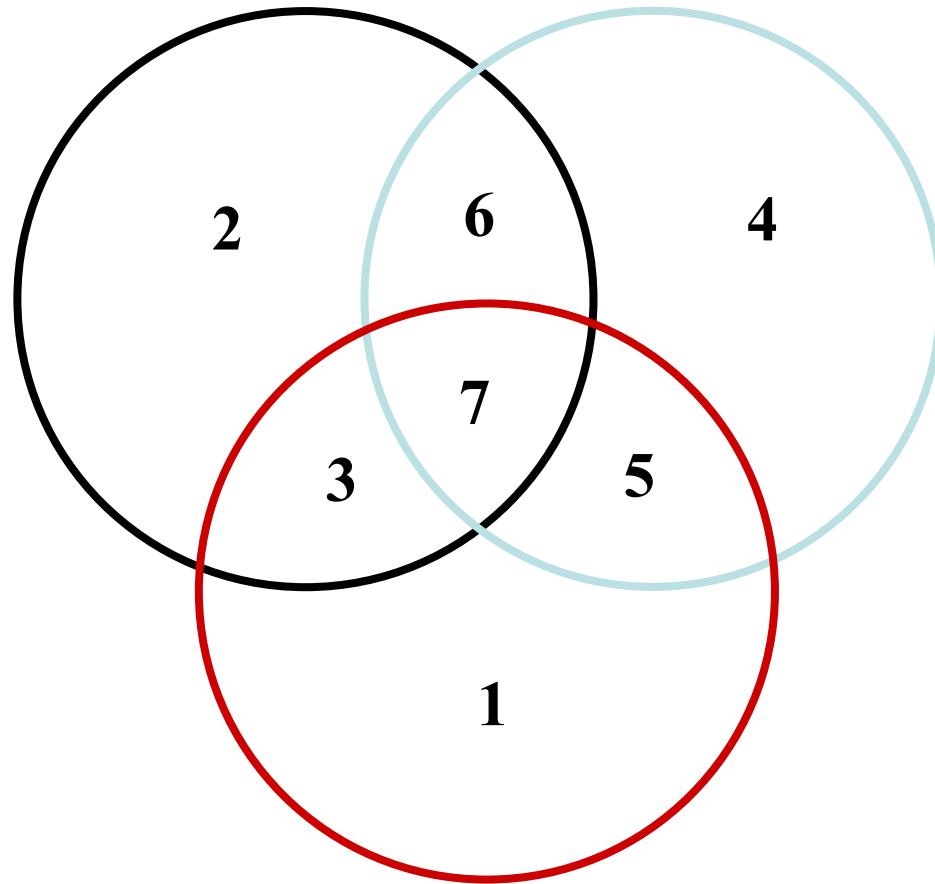
Error Detection

- Why we need it ?
 - To **avoid retransmission** of whole packet or message
- What to do if error detected ?
 - **Discard**, and request a new copy of the frame:
 - explicitly or implicitly
 - Try to **correct** error, if possible

Error Detection

- Validates correctness of each frame
- Errors **checked at many levels**
- Demodulation of signals into symbols (analog)
- Bit error detection/correction (digital)—our main focus
 - Within network adapter (CRC check)
 - Within IP layer (IP checksum)
 - Possibly within application as well

Error Detection and Correction



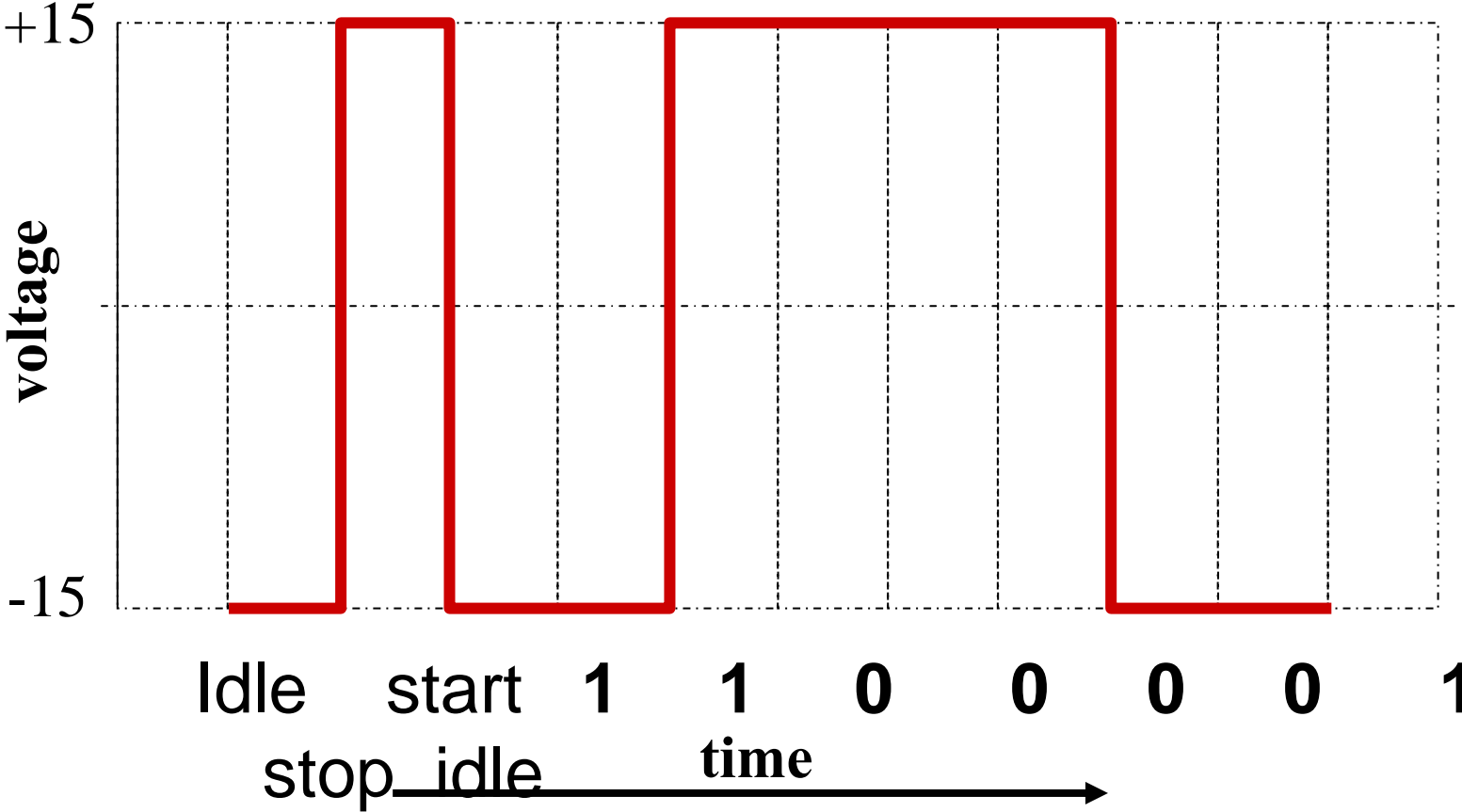
Error Detection

- Analog errors
 - Example of signal distortion
 - Discuss to illustrate input to digital level
- Hamming distance
 - Parity and voting
 - Concept and usefulness
 - Hamming codes
- Errors bits or error bursts
- Digital error detection techniques: two-dimensional parity, checksum, CRC

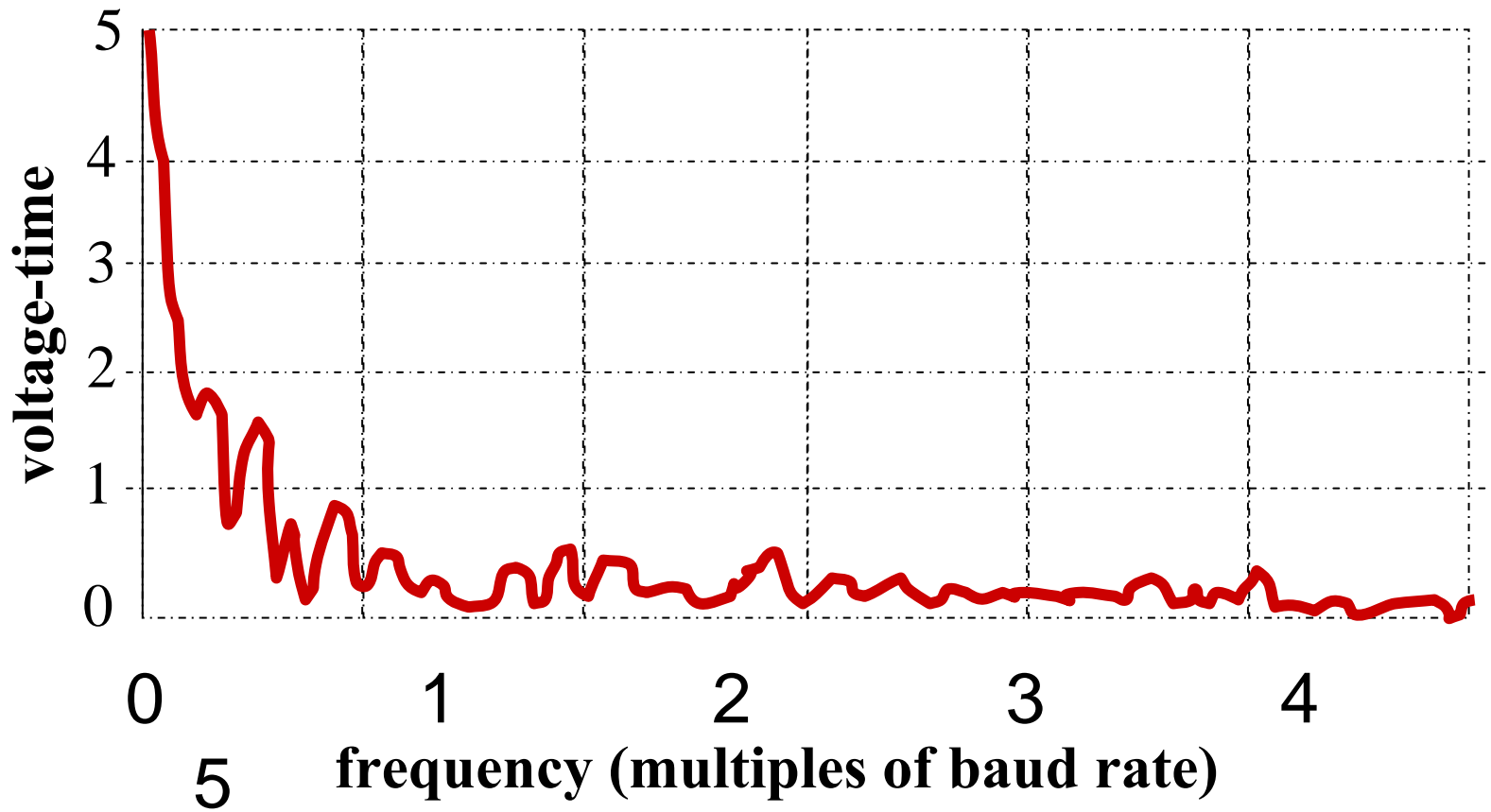
Analog Errors – Signal Distortion

- Consider RS-232 encoding of character 'Q'
- Assume idle wire (-15V) before and after signal
- Calculate **frequency distribution** of signal $A(f)$ using a Fourier transform
- Apply **low-pass filter** (drop high frequency components)
- Calculate signal using inverse Fourier

RS-232 Encoding of "Q"

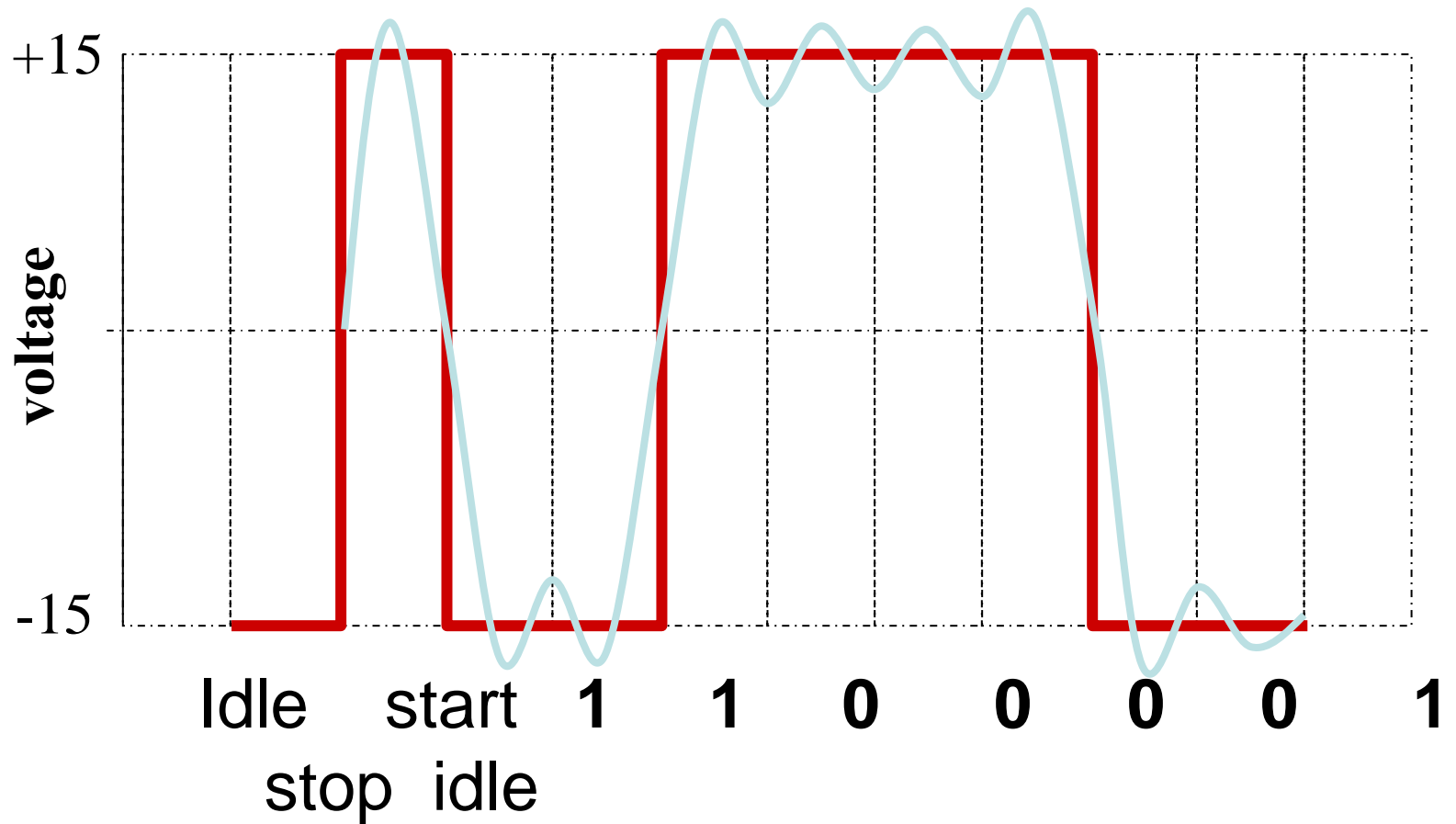


Frequency Distribution of 'Q' Encoding



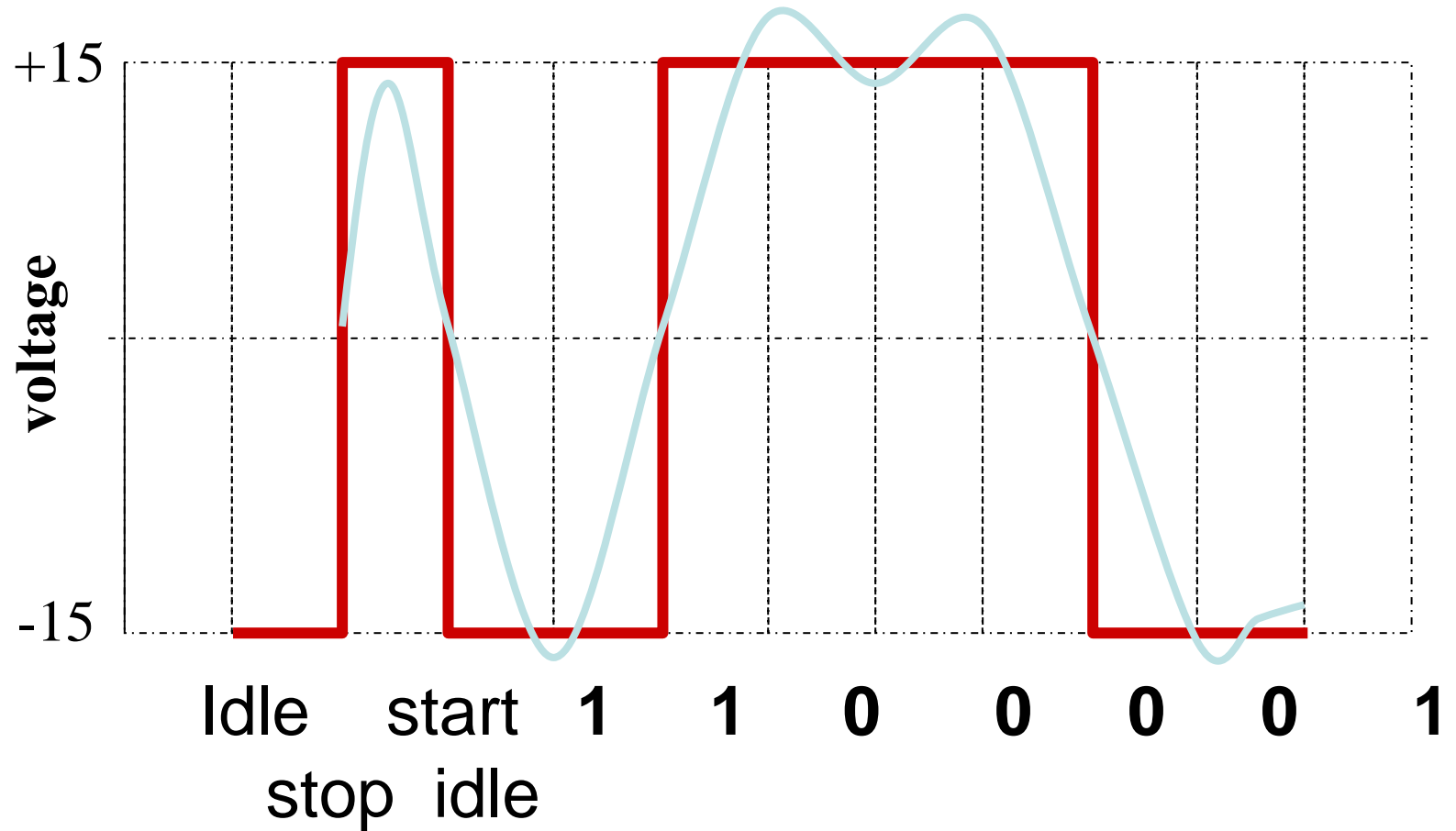
Limited-Frequency Signal Response

(bandwidth = baud rate)

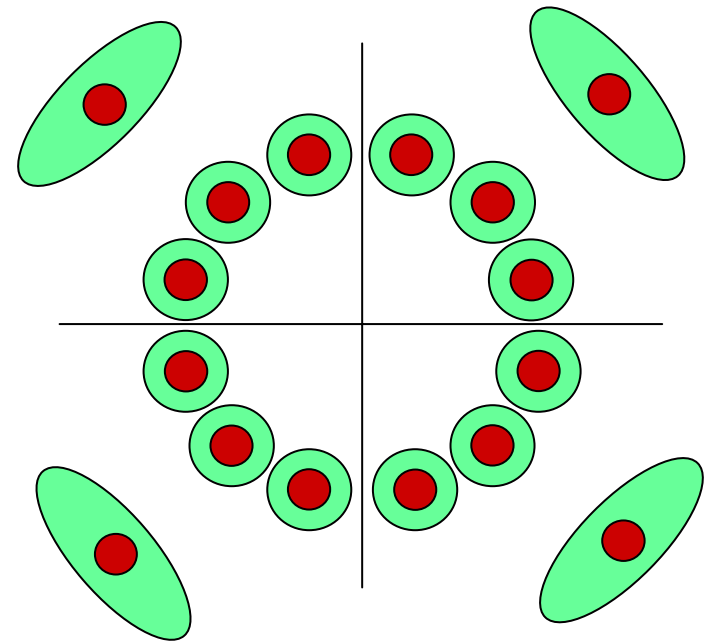
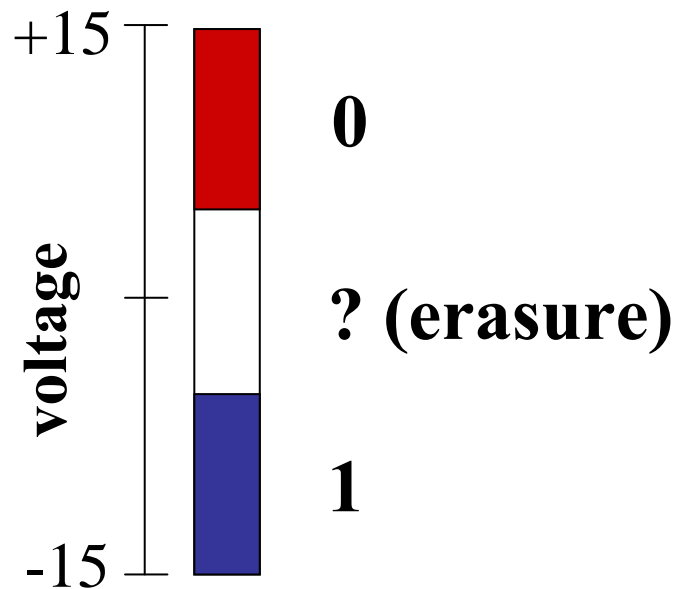


Limited-Frequency Signal Response

(bandwidth = baud rate/2)



Error Detection and Correction



- possible binary voltage encoding symbol
 - possible QAM symbol
 - neighborhoods in green
 - all other space results in erasure
- neighborhood erasure region **Input to digital level: valid symbols or erasures**

Error Detection: How ?

- How to detect error ?
 - **Add redundant information** to a frame to determine errors
- Transmit two complete **copies** of data
 - n redundant bits for n -bit message
 - Error at the same position in two copies go undetected

Error Detection: How ?

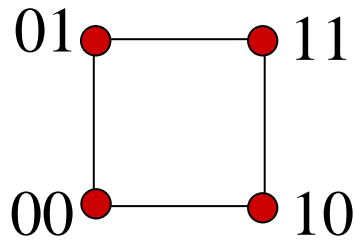
- We want only k redundant bits for an n -bit message, where $k \ll n$
 - In Ethernet, 32-bit CRC for 12,000 bits (1500 bytes)
- k bits are **derived** from the original message
- Both the sender and receiver know the **algorithm**

Hamming Distance

- 1-bit error-detection with parity
- 1-bit error-correction with voting
- 2-bit erasure-correction with voting
- Definition and bounds
- Hamming codes (1-bit error correction)

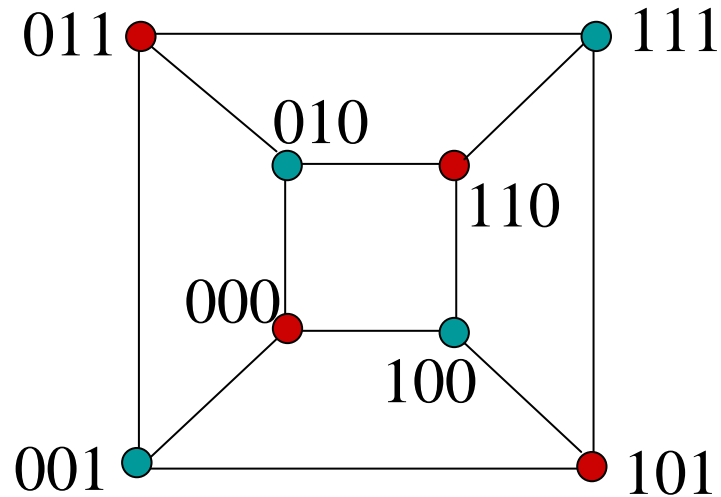
1-bit Error Detection with Parity

- Every code has **even number of 1's**



**Valid
codes**

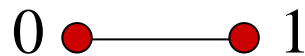
**if only 1 bit flips,
it can be detected**



**Parity encoding:
gray dots are invalid
and indicate errors**

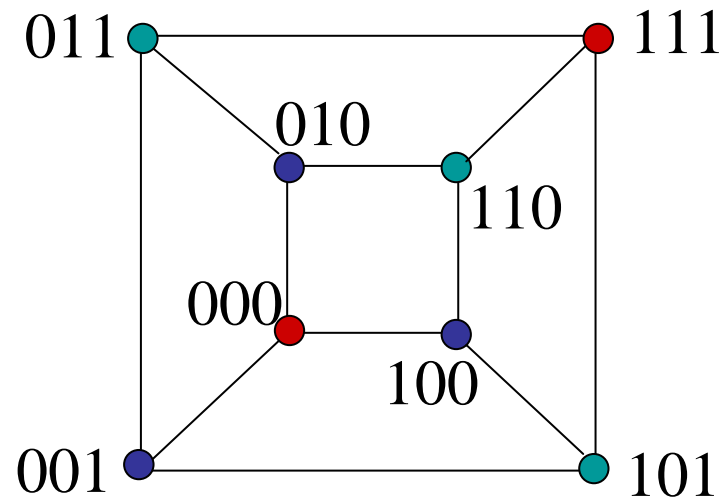
1-bit Error Correction with Voting

- Every code is **copied** three times



**Valid
codes**

- if only 1 bit flips, it can be corrected
- even with 2 erasures, bit can be recovered



**gray dots correct to “1”
blue dots correct to “0”**

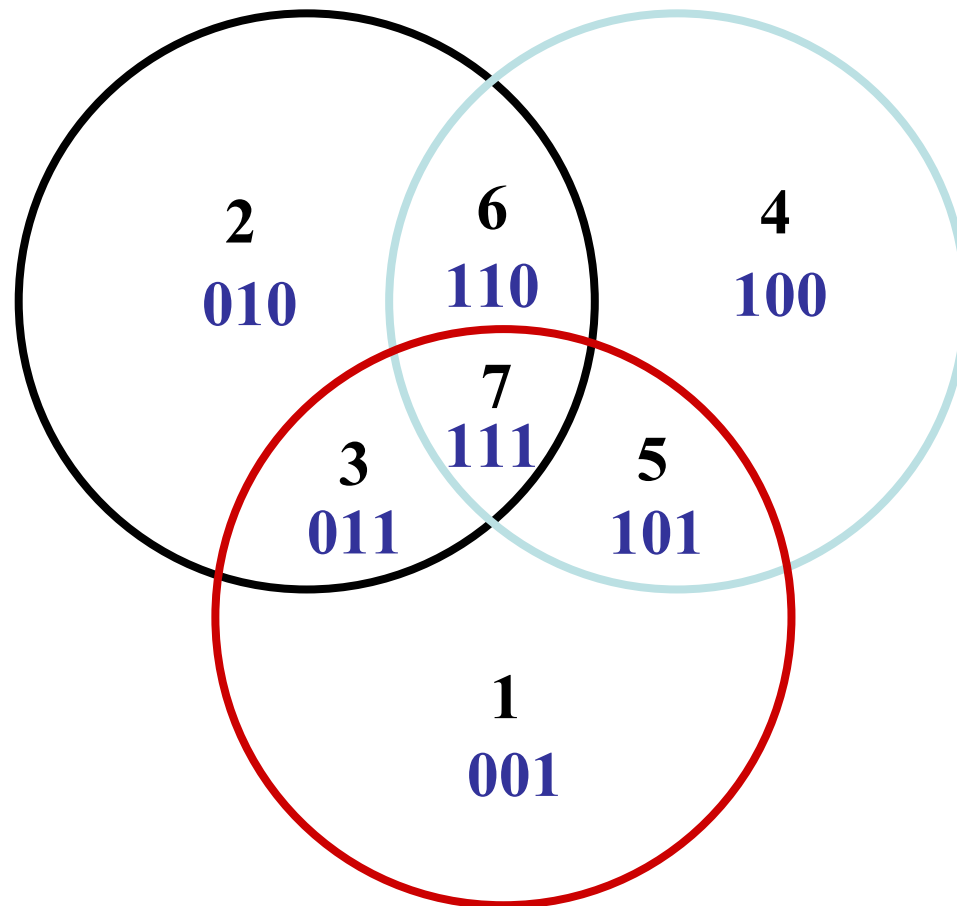
Hamming Distance (1950 Paper)

- Minimum number of bit flips between code words
 - 2 flips for parity
 - 3 flips for voting
- n-bit error detection
 - No code word changed into another code word
 - Requires Hamming distance of $n+1$
- n-bit error correction
 - N-bit neighborhood: all code words within n bit flips
 - No overlap between n-bit neighborhoods
 - Requires Hamming distance of $2n+1$

Hamming Codes (1950 Paper)

- Construction for 1-bit error-correcting codes
- Minimal number of **check bits** required
- Construction
 - Number bits from 1 upward
 - Powers of 2 are check bits
 - All others are data bits
 - Check bit j is XOR of all bits k such that $(j \text{ AND } k) = j$
- – Example: 4 bits of data, 3 check bits

Error Detection and Correction



Error Bits or Error Burst

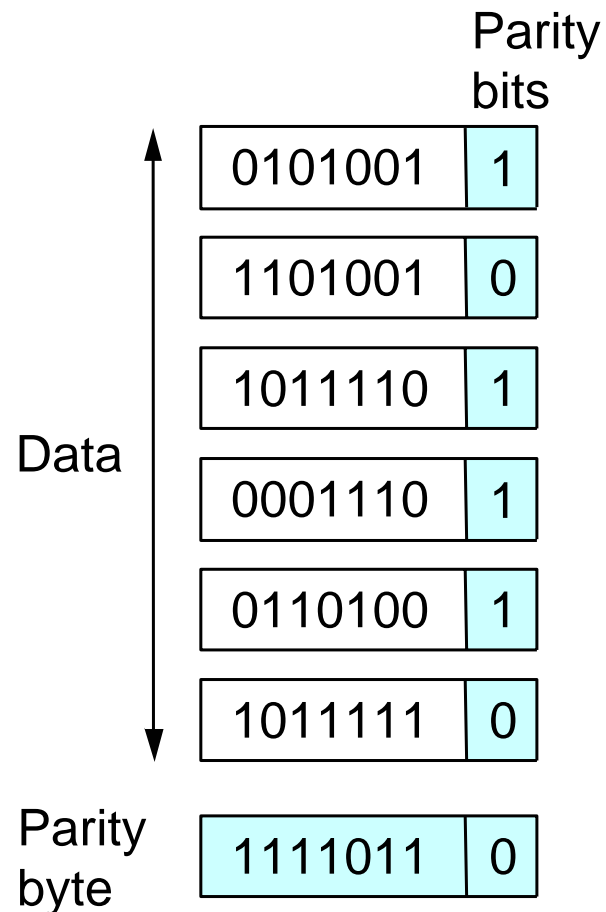
- Common model of errors
 - Probability of error per bit
 - Error in each bit **independent** of others
 - Value of incorrect bit independent of others
- Burst model
 - Probability of back-to-back bit errors
 - Error probability dependent on adjacent bits
 - Value of errors may have structure
- Why assume bursts?
 - Appropriate for some media (e.g., Radio)
 - Faster signaling rate enhances such phenomena

Digital Error Detection Techniques

- Two-dimensional parity
 - Detects up to 3-bit errors
 - Good for burst errors
- Internet checksum (used as backup to CRC)
 - Simple addition
 - Simple in software
- Cyclic redundancy check (**CRC**)
 - Powerful mathematics
 - Tricky in software, simple in hardware
 - Used in network adapter

Two-Dimensional Parity

- Adding one extra bit to a 7-bit code to balance 1s
- extra parity byte for the entire frame
- Catches all 1-, 2- and 3-bit errors and most 4-bit errors
- 14 redundant bits for a 42-bit message, in the example



Two-Dimensional Parity

0	1	0	0	0	1	1	1
0	1	1	0	0	0	1	1
0	1	1	0	1	1	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	1	1
0	1	1	0	1	1	0	0
1	1	0	1	0	1	0	0

Internet Checksum Algorithm

- Not used at the link level but provides same sort of functionality as CRC and parity
- Idea:
 - Add up all words (16-bit integers) that are transmitted
 - Transmit the result (checksum) of that sum
 - Receiver performs the same calculation on received data and compares the result with the received checksum
 - If the results do not match, an error is detected
- **16 redundant bits** for a message of any length
- **Weak protection**, accepted as a last line of defense

Internet Checksum Algorithm

View message as a sequence of 16-bit integers; sum using 16-bit **ones-complement arithmetic**; take ones-complement of the result.

- ```
u_short cksum(u_short *buf, int count)
{
 register u_long sum = 0;
 while (count--)
 {
 sum += *buf++;
 if (sum & 0xFFFF0000)
 {
 /* carry occurred, so wrap around */
 sum &= 0xFFFF;
 sum++;
 }
 }
 return ~(sum & 0xFFFF);
}
```

# Cyclic Redundancy Check

## Theory

- based on finite-field (binary-valued) **arithmetic**
- bit string represented as **polynomial**
- **coefficients** are binary-valued
- divide bit string polynomial by **generator polynomial** to generate CRC

## Practice

- bitwise XOR's

# Cyclic Redundancy Check

- Add  **$k$  bits** of redundant data to an  $n$ -bit message
  - want  $k \ll n$
  - e.g.,  $k = 32$  and  $n = 12,000$  (1500 bytes)
- Represent  $n$ -bit message as  $n-1$  degree polynomial
  - e.g., MSG=10011010 as  $M(x) = x^7 + x^4 + x^3 + x^1$
  - Sender and receiver exchange polynomials
- Let  $k$  be the degree of some agreed-upon divisor/ generator polynomial
  - e.g.,  $C(x) = x^3 + x^2 + 1$

# Cyclic Redundancy Check

- Transmit polynomial  $P(x)$  that is **evenly divisible** by  $C(x)$ 
  - shift left  $k$  bits, i.e.,  $M(x)x^k$
  - add remainder of  $M(x)x^k / C(x)$  into  $M(x)x^k$
- Receiver receives polynomial  $P(x) + E(x)$ 
  - $E(x) = 0$  implies no errors
- Receiver divides  $(P(x) + E(x))$  by  $C(x)$ ; remainder will be zero **ONLY** if:
  - $E(x)$  was zero (no error), or
  - $E(x)$  is exactly divisible by  $C(x)$

# CRC Example - Sender

- $C(x) = x^3 + x^2 + 1 = 1101 \rightarrow$  generator
- $M(x) = x^8 + x^6 + x^5 + x^4 + 1 = 101110001 \rightarrow$  message

$$\begin{array}{r} 1101 \quad ) \overline{101110001000} \\ \quad \underline{1101} \\ \quad \quad 1101 \\ \quad \quad \underline{1101} \\ \quad \quad \quad 1101 \\ \quad \quad \quad \underline{1101} \\ \quad \quad \quad \quad 00001000 \\ \quad \quad \quad \quad \quad \underline{1101} \\ \quad \quad \quad \quad \quad \quad 101 \text{ remainder} \end{array}$$

# CRC Example - Receiver

- $C(x) = x^3 + x^2 + 1 = 1101 \rightarrow$  generator
- $M(x) = x^{11} + x^9 + x^8 + x^7 + x^3 + x^2 + 1 = 101110001101 \rightarrow$  message

$$\begin{array}{r} \hline 1101 \quad ) \quad 101110001101 \\ \underline{1101} \phantom{00000000000} \\ 1101 \phantom{0000000000} \\ \underline{1101} \phantom{000000000} \\ 00001101 \phantom{00000} \\ \underline{1101} \phantom{00000} \\ 0 \text{ correct !} \end{array}$$

# CRC Example - Receiver

- $C(x) = x^3 + x^2 + 1 = 1101 \rightarrow$  generator
- $M(x) = x^{11} + x^9 + x^8 + x^7 + x^3 + x^2 + 1 = 101110001101 \rightarrow$  message

1101 ) 101101001101

1101

1100

1101

1100

1101

1110

1101

111 incorrect !

Two bits are flipped

# CRC Example - Receiver

- $C(x) = x^3 + x^2 + 1 = 1101 \rightarrow$  generator
- $M(x) = x^{11} + x^9 + x^8 + x^7 + x^3 + x^2 + 1 = 101110001101 \rightarrow$  message

$$\begin{array}{r}
 \text{1101} \quad ) \quad \text{101101011101} \\
 \underline{\text{1101}} \\
 \text{1100} \\
 \underline{\text{1101}} \\
 \text{1101} \\
 \underline{\text{1101}} \\
 \text{1101} \\
 \underline{\text{1101}} \\
 \text{1101} \\
 \underline{\text{1101}} \\
 \text{0}
 \end{array}$$

Three bits are flipped

0 **incorrectly correct !!!**

# Selecting $C(x)$ Non-divisible by $E(x)$

- All single-bit errors, as long as the  **$x^k$  and  $x^0$  terms** have non-zero coefficients.
- All double-bit errors, as long as  $C(x)$  contains a factor with **at least three terms**
- Any odd number of errors, as long as  $C(x)$  contains the factor  **$(x + 1)$**
- Any 'burst' error (i.e., sequence of error bits) for which the length of the **burst is less than  $k$  bits**
- Most burst errors of larger than  $k$  bits can also be detected
- See Table 2.6 on page 96 for common  $C(x)$

# Error Detection or Correction ?

- Detection implies **discarding** message and **waiting** for retransmission
  - uses bandwidth
  - introduces latency
- Error correction requires more redundant bits to send all the time : error-correcting code (FEC)
- Error correction is useful when:
  - errors are quite probable (wireless links)
  - retransmission cost is too high (latency in satellite link, multicast)

# Topics Already Covered

- Elements of networks: nodes and links
- Building a packet abstraction on a point-to-point link
  - Transmission methods and challenges
  - Limiting factors on data rates
  - Defining units of communication data
  - Detecting transmission errors
- Next: simulating an error-free channel

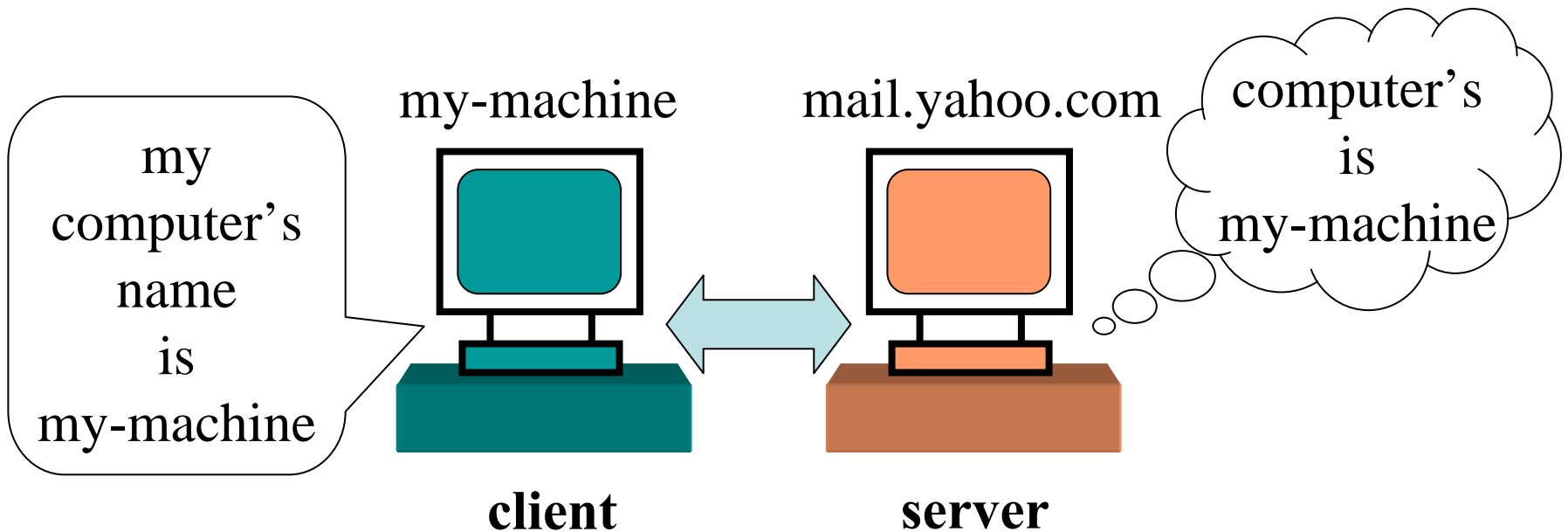
# Reliable Transmission

# Point-to-Point Links

- Reading: Peterson and Davie, Ch. 2
- Hardware building blocks
- Encoding
- Framing
- Error Detection
- **Reliable transmission**
  - **Sliding Window Algorithm**

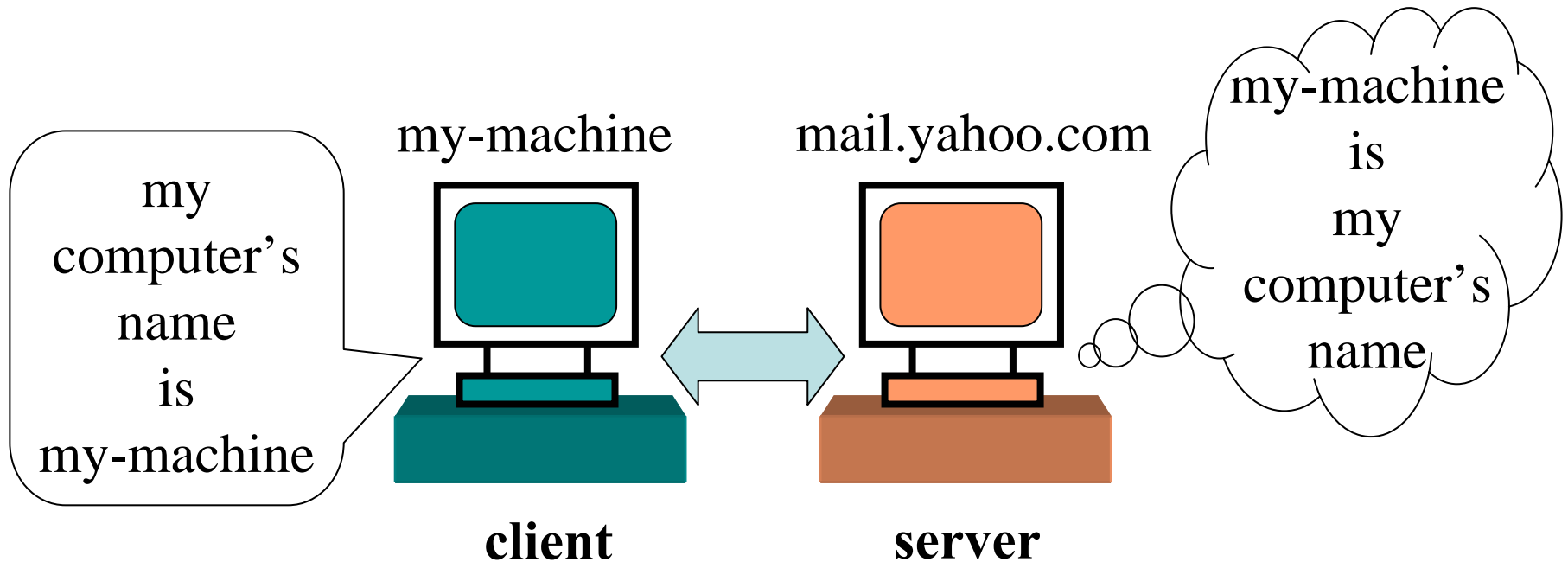
# Reliable Transmission

- Higher level of abstraction (transport layer vs. data link layer)



# Reliable Transmission

- Higher level of abstraction (transport layer vs. data link layer)



# Reliable Transmission

- Error-correcting codes are **not advanced** enough to handle the range of bit and burst errors
  - Corrupt frames generally must be discarded
  - A reliable link-level protocol must recover from discarded frames
- Goals for reliable transmission
  - Make channel appear reliable
  - Maintain packet order (usually)
  - Impose low overhead / allow full use of link

# Reliable Transmission

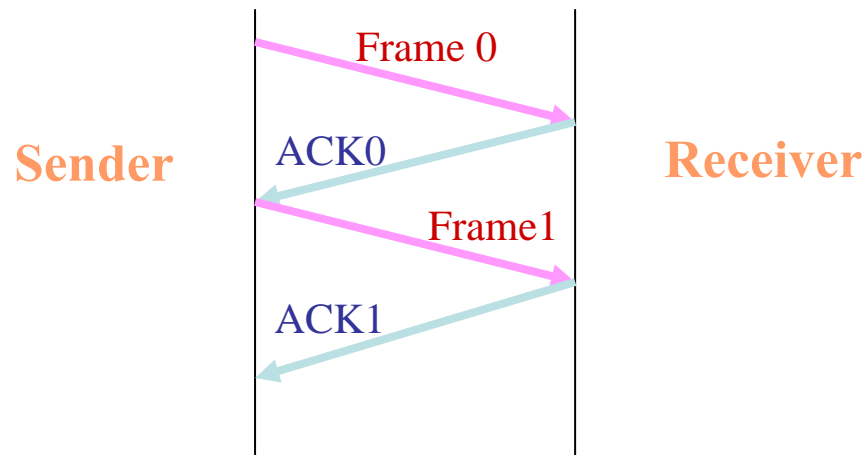
- Reliability accomplished using **acknowledgments and timeouts**
  - ACK is a small **control frame** confirming reception of an earlier frame
  - Having no ACK, sender retransmits after a timeout
- Automatic Repeat reQuest (**ARQ**) algorithms
  - Stop-and-wait
  - Concurrent logical channels
  - Sliding window
    - Go-back-n, or selective repeat
- Alternative: forward error correction (FEC)

# Automatic Repeat reQuest

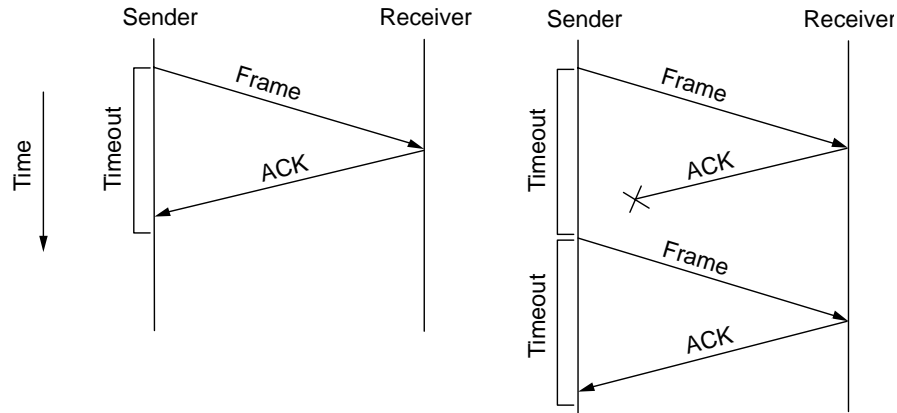
- Acknowledgement (ACK)
  - Receiver tells sender when frame received
  - Cumulative **ACK** (used by TCP): have received specified frame and all previous
  - Selective ACK (**SACK**): specifies set of frames received
  - Negative ACK (**NACK** or NAK): receiver refuses to accept frame now, e. g. , when out of buffer space
- Timeout: sender decides that frame was lost and **tries again**
- ARQ also called Positive

# Stop-and-Wait

- Send a single frame
- Wait for ACK or timeout
  - If ACK received, continue with next frame
  - If timeout occurred, send again (and wait)
    - Frame lost in transit; or corrupted and discarded

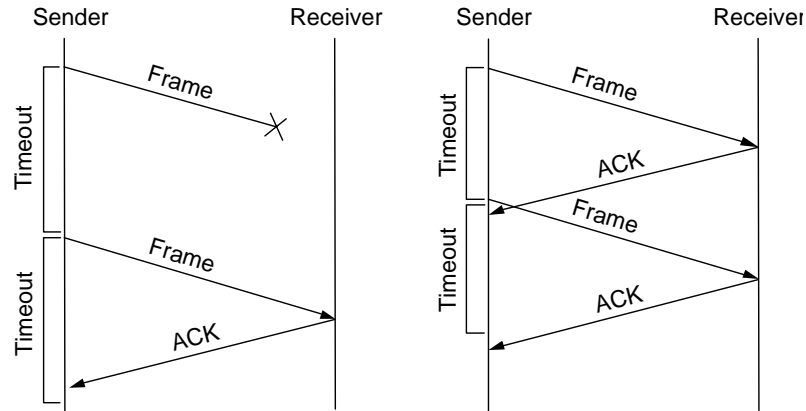


# Acknowledgments and Timeouts



(a)

(c)



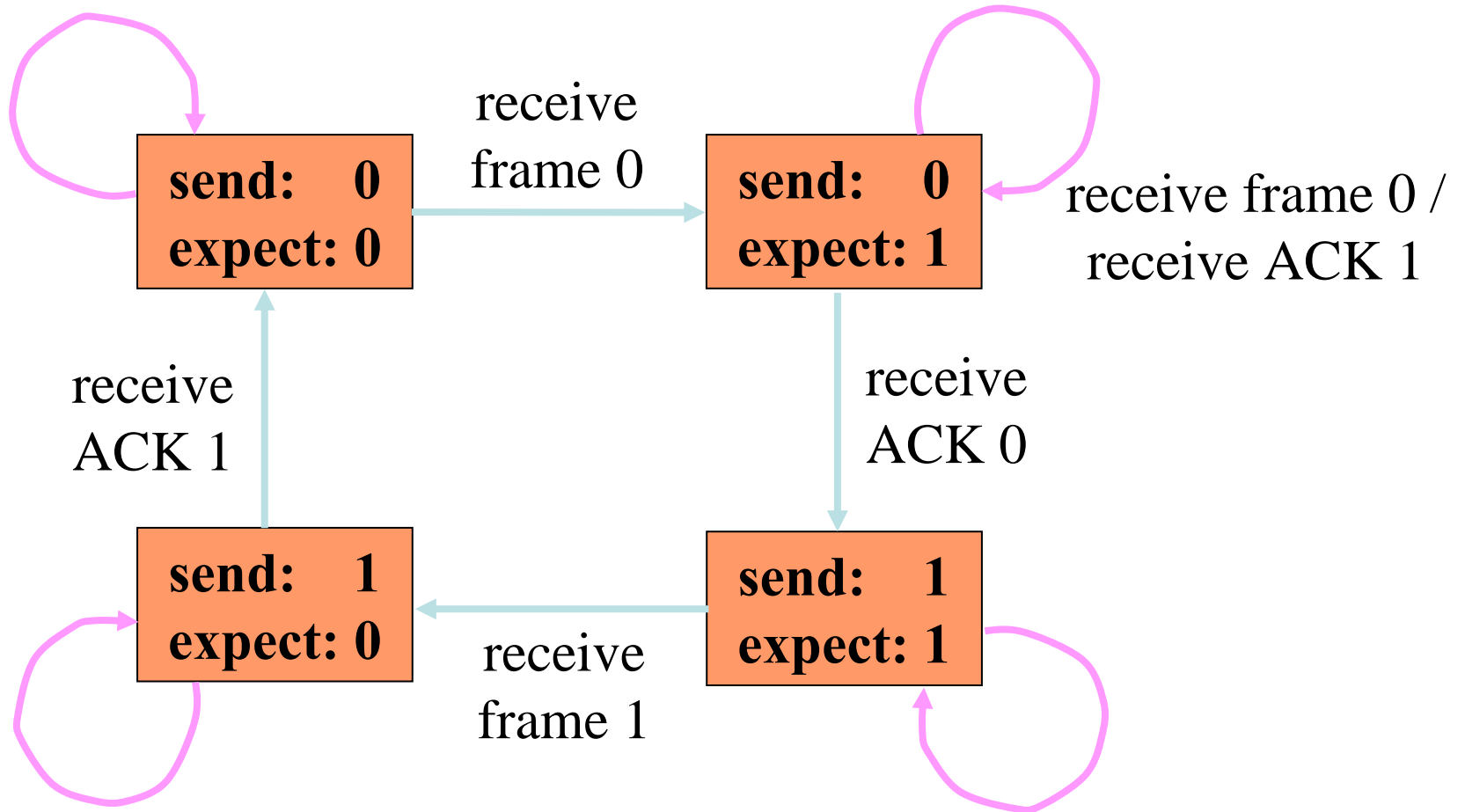
(b)

(d)

# Stop-and-Wait

- If receiver receives a **frame correctly**, but sender receives the **ACK after timeout** ...
  - Sender resends the frame; how the receiver knows it's the **same** frame or the **next** frame ?
- Requires **frame identification**
  - Duplicate frame ?
  - Duplicate ACK ?
  - **1 bit is enough** (if physical network maintains order)
    - sender tracks frame ID to send
    - receiver tracks next frame ID expected

# Stop-and-Wait State Diagram



# Stop-and-Wait

- Frames delivered **reliably and in order**
- Is that enough ?
  - No, we need performance, too.
- Problem: **keeping the pipe full ... ?**
- Example
  - 1.5Mbps link x 45ms RTT = 67.5Kb (~8KB)
  - 1KB frames implies 182 Kbps (1/8th link utilization)
  - Want the sender to transmit 8 frames before waiting for ACK
  - Throughput remains 182 Kbps **regardless of the link bandwidth !!**

# Concurrent Logical Channels

- **Multiplex** several logical channels over a single p-to-p physical link (include channel ID in header)
- Use **stop-and-wait** for each logical channel
- Maintain three bits of **state** for each logical channel:
  - Boolean saying whether the channel is currently busy
  - Sequence number for frames sent on this channel
  - Next sequence number to expect on this channel
- ARPANET IMP-IMP supported 8 logical channels over each ground link (16 over each satellite link)

# Concurrent Logical Channels

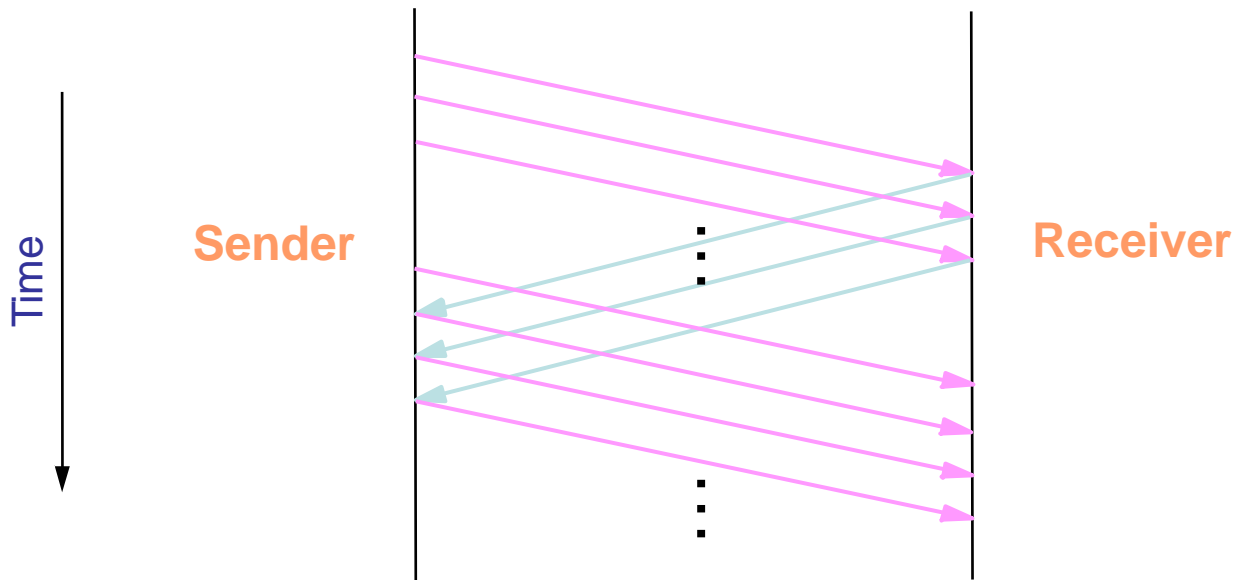
- **Header** for each frame include 3-bit channel number and 1-bit sequence number
  - Same number of bits (4) as the sliding window requires to support up to 8 outstanding frames on the link
- **Characteristics**
  - Separates reliability from flow control and frame order
  - Each channel limited by stop-and-wait bandwidth
  - Aggregate bandwidth uses full physical link
  - Supports multiple communicating processes
  - Can use more than one channel per process
    - But no frame ordering between channels

# Approaches for Reliable Transmission ...

- Stop- and- wait
  - Provides reliable, in-order delivery
  - Sacrifices performance
- Multiple logical channels
  - Provides reliable delivery at full link bandwidth
  - Sacrifices packet ordering
- Sliding window: meets all three goals

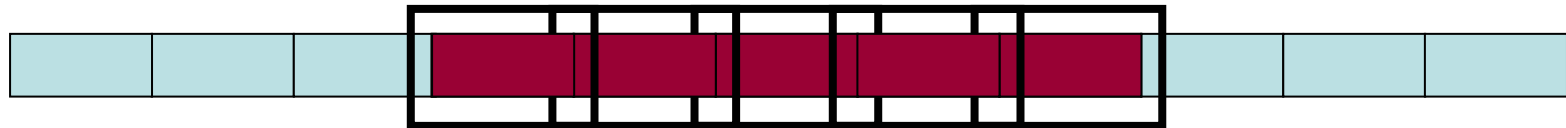
# Sliding Window

- Allow sender to transmit **multiple frames** before receiving an ACK, thereby keeping the pipe full
- **Upper bound** on outstanding un-ACKed frames
- Also used at the transport layer (by TCP)



# Sliding Window Concepts

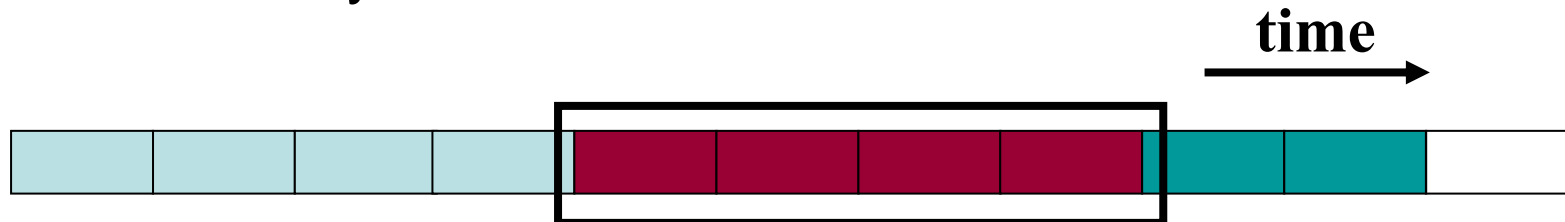
- consider **ordered stream** of data
  - broken into frames
  - stop-and-wait
    - window of one frame
    - slides along stream over time



- sliding window algorithms generalize this notion
  - multiple-frame send window
  - multiple-frame receive window

# Sliding Window Concepts

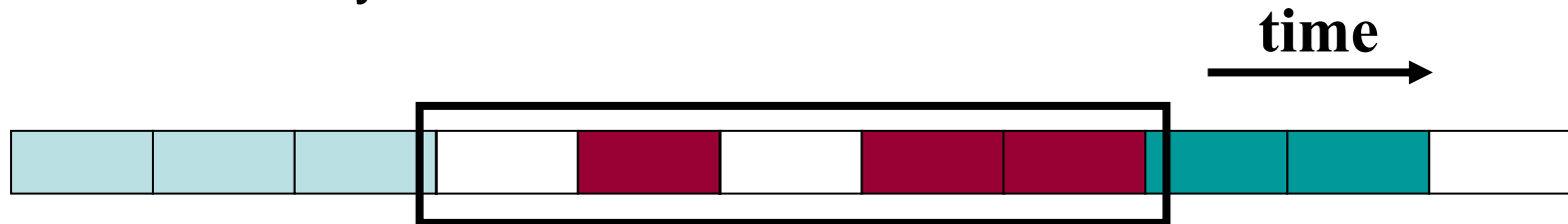
- send window
  - fixed length, containing numbered frames
  - starts at **earliest unacknowledged** frame
  - only frames in window sent over network



- **Green**: sent and acknowledged
- **Red**: sent (or can be sent) but not acknowledged
- **Blue**: available, but not within send window

# Sliding Window Concepts

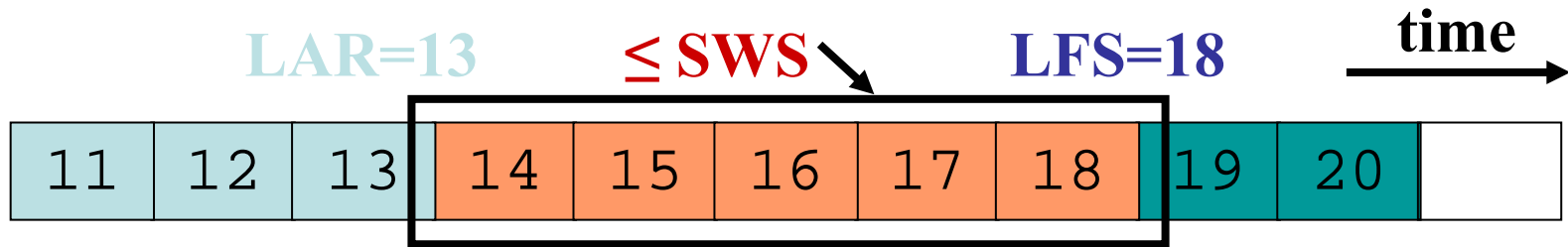
- receive window
  - fixed length (unrelated to send window)
  - starts at **earliest unreceived** frame
  - only frames in window are buffered



- **Green**: received and delivered
- **Red**: received and buffered
- **Blue**: received and discarded

# Sliding Window - Sender

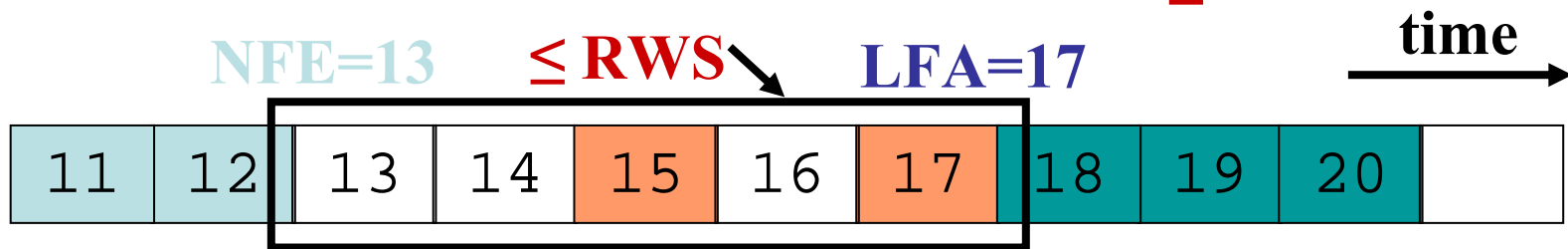
- Assign sequence number to each frame (**SeqNum**)
- Maintain three state variables:
  - send window size (**SWS**)
  - last acknowledgment received (**LAR**)
  - last frame sent (**LFS**)
- Maintain invariant: **LFS - LAR ≤ SWS**



- **Advance LAR** when ACK arrives
- Buffer up to **SWS** frames and associate timeouts

# Sliding Window - Receiver

- Maintain three state variables
  - receive window size (**RWS**)
  - largest frame acceptable (**LFA**)
  - next frame expected (**NFE**)
- Maintain invariant: **LFA - NFE + 1 ≤ RWS**



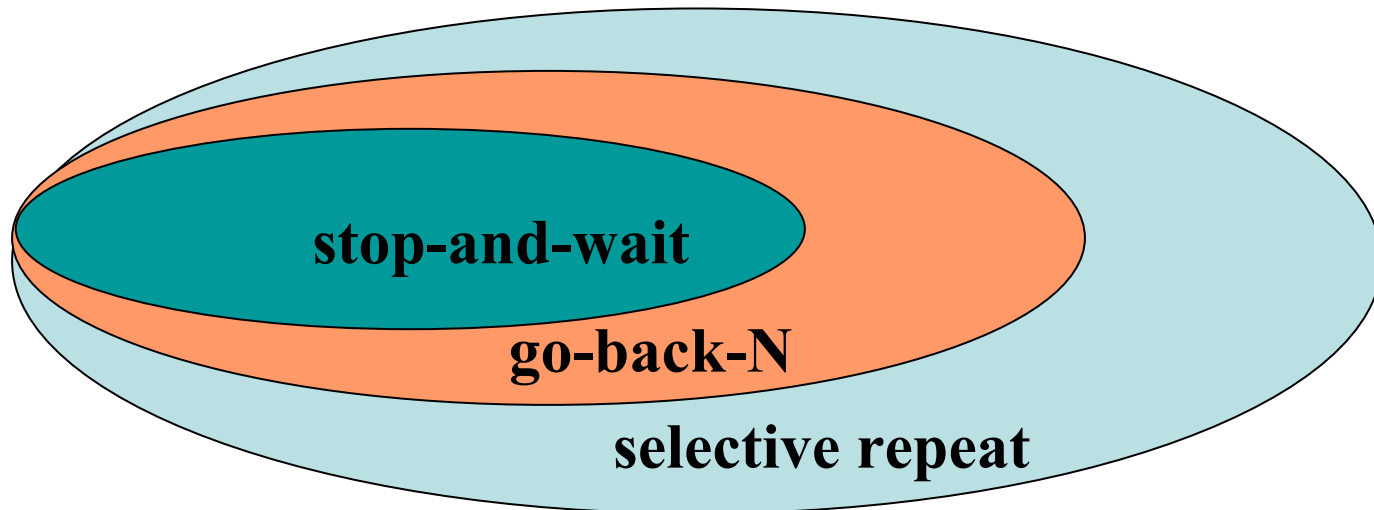
- Frame `seqNum` arrives:
  - if **NFE ≤ SeqNum ≤ LFA** → accept
  - if **SeqNum ≤ NFE** or **SeqNum > LFA** → discarded
- Send cumulative ACKs

# Sliding Window Issues

- When a timeout occurs, data in transit **decreases**
  - Pipe is no longer full when packet losses occur
  - Problem aggravates with delay in packet loss detection
- **Early detection** of packet losses improves performance:
  - Negative Acknowledgements (NACKs)
  - Duplicate Acknowledgements
  - Selective Acknowledgements (SACKs)
    - Adds complexity but helps keeping the pipe full

# Sliding Window Classification

- Stop-and-wait:  $SWS=1, RWS=1$
- Go-back-N:  $SWS=N, RWS=1$
- Selective repeat:  $SWS=N, RWS=M$   
(usually  $M = N$ )



# Sliding Window: Go-back-N

- Go-back-N (**SWS=N, RWS=1**)
- Receiver only buffers 1 frame
- If frame lost, sender may need to resend N frames
  - *i. e.* , sender goes back N frames
- Variations
  - How long is the frame timeout?
  - Does receiver send NACK for out-of-sequence frame?

# Sliding Window: Selective Repeat

- Selective repeat (**SWS=N, RWS=M**)
- Receiver buffers M frames
- If frame lost, sender must resend only
  - frames lost within receive window
  - frames past end of receive window
- Variations
  - How long is the frame timeout?
  - Use cumulative or per-frame ACK?
  - Does protocol adapt timeouts?
  - Does protocol adapt SWS and/or RWS?

# Sequence Number Space

- **SeqNum** field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames (**SWS**)
- **$SWS \leq \text{MaxSeqNum} - 1$**  is not sufficient
  - suppose 3-bit **SeqNum** field (0..7); **SWS=RWS=7**
  - sender transmits frames 0..6; which arrive successfully (receiver window advances)
  - ACKs are lost; sender retransmits 0..6
  - receiver expecting 7, 0..5, but receives second incarnation of 0..5 assuming them as 8<sup>th</sup> to 13<sup>th</sup> frame

# Required Sequence Number Space ?

- Assume  $SWS=RWS$  (simplest, and typical)
  - Sender transmits full SWS
  - Two extreme cases at receiver
    - None received (waiting for  $0\dots SWS-1$ )
    - All received (waiting for  $SWS\dots 2*SWS-1$ )
- All possible packets must have unique `SeqNum`
- $SWS < (MaxSeqNum+1) / 2$  or  $SWS+RWS < MaxSeqNum+1$  is the correct rule
- Intuitively, `SeqNum` “slides” between two halves of sequence number space

# Sliding Window Assumptions

- Sliding window protocol leads to in-order delivery of all frames, with the following assumptions
  - Frames can be **delayed** an arbitrary but finite amount of time
  - Frames can be **lost**
  - Frames can arrive with **detectable errors**
  - Frames cannot arrive with undetectable errors
  - Frames **arrive in the order** sent (possibly with errors)
- Are these assumptions adequate?

# Sliding Window Correctness

- need one more assumption:
  - Any given frame is received without errors after a **finite number of retransmissions**
- proof in two steps
  - establish correctness assuming infinite sequence number space
  - show that finite sequence number space does not affect result as long as it has  $\geq 2 \max(SWS, RWS)$  possible numbers

# Separation of Concerns

- Carefully distinguish different **functions rolled together in one mechanism**
  - Each function is necessary here ?
  - Each function is supported in best effective way ?
- Sliding window combines
  - Reliable delivery
  - Ordered delivery
  - Flow control
- Is it the **right thing to do at link level** ?