Data and Computer Communications

Chapter 2 – Data Link Control Protocols

Data Link Control Protocols

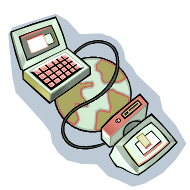
"Great and enlightened one," said Ten-teh, as soon as his stupor was lifted, "has this person delivered his message competently, for his mind was still a seared vision of snow and sand and perchance his tongue has stumbled?"

"Bend your ears to the wall," replied the Emperor, "and be assured."

—Kai Lung's Golden Hours, Earnest Bramah

Data Link Control Protocols

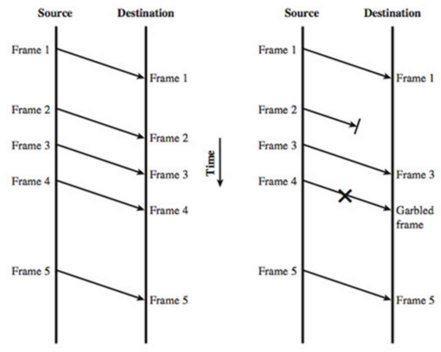
- when sending data, to achieve control, a layer of logic is added above the Physical layer
  - data link control or a data link control protocol
- to manage exchange of data over a link midium:
  - frame synchronization
  - flow control
  - error control
  - addressing
  - control and data
  - link management



Flow Control

- ensure sending entity does not overwhelm receiving entity
  - prevent buffer overflow
- influenced by:
  - transmission time
    - time taken to emit all bits into medium
  - propagation time
    - time for a bit to traverse the link
- assumption is all frames are successfully received with no frames lost or arriving with errors

Model of Frame Transmission



(a) Error-free transmission

(b) Transmission with losses and errors

Stop and Wait

- simplest form of flow control

source transmits frame

destination receives frame and replies with acknowledgement (ACK)

source waits for ACK before sending next frame

destination can stop flow by not sending ACK

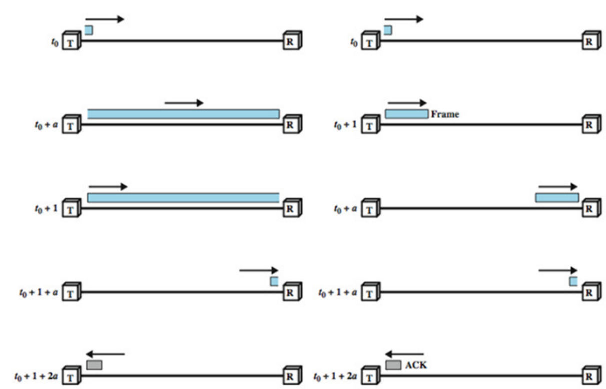
- works well for a message sent in a few large frames
  - stop and wait becomes inadequate if large block of data is split into small frames by source

How to Stop and Wait

- With the use of multiple frames for a single message, the stop-and-wait procedure may be inadequate, mainly since only one frame at a time can be in transit. Assume:
  - F is the length of a frame – in bits.
  - B is the bit length of a link (number of bits to fill-up a given link)
  - L is the length of the link
- Since B is the maximum number of bits present on the link at an instance in time when a stream of bits fully occupies the link.
- In situations where $B > F$, serious inefficiencies result, as shown in the following slide. In the figure, the transmission time (the time it takes for a station to transmit a frame) is normalized to one, and the propagation delay (the time it takes for a bit to travel from sender to receiver) is expressed as the variable $a = B / L$.

If given particular values for B/L/F can you
- Calculate link utilization?
- Optimal frame size?

Stop and Wait Link Utilization



(a) $a < 1$ (b) $a > 1$

Types of Delays

There are four types of delay in digital networks:

Processing delays, Transmission delay, Propagation delay, and Queuing delay.

Processing Delay (D_{proc}):
is the delay due to the processing of information within network elements, such as the processing of packet header, error check, etc…

- D_{proc} is a fixed number for per packet (μsec)
- D_{proc} depends on the elements' processing power

Most network elements use multi-processors so the frame processing time takes place while packets are waiting on the transmission queue.

Transmission Delay (D_T)

Transmission Delay (D_T):
is the delay consumed by the transmitter to put the frame into the medium:

$$D_T = \frac{Number\ of\ bits\ in\ msg\ (bits)}{bit\ rate\ of\ transmisison\ link\ (bps)} = \frac{F}{T}$$

e.g. To send a file of 1Mb on a 500kbps wireless transmitter, the transmission delay should be 2s

- Transmission delay depends on the frame size when bitrate is fixed. Henceforth, for the same link, D_T is a function of frame size.

Propagation Delay (D_P)

Propagation Delay (D_P):
is the delay of the electromagnetic wave propagating through the link media:

$$D_P = \frac{distance\ (L\ in\ Km)}{Speed\ of\ electromagnetic\ wave\ (^{Km}/_{\mu sec})}$$

e.g. Assume the distance between the wireless transmitter and receiver being 30Km. The electromagnetic wave propagates at the speed of light in free space. Then the propagation delay will be 30000/300000000=0.0001s

- Propagation delay is fixed for the same link since the length and electromagnetic wave speed are typically fixed for the same link.

Queuing Delay (D_Q)

Queuing Delay (D_Q):
is the delay of the frame awaiting in the memory of the transmission queue before being transmitted. Queuing delay may be caused by many reasons, including frame priority difference, random back-off due to collision, protocol operations (such as handshake), etc.:

- Queuing delay is dependant on the network element memory size and its interface capabilities
- Queuing delays may occur at the receiver if overwhelmed by frames
- Queuing delay could be zero if the network is lightly loaded and might grow rapidly as the network become congested.

Link Measure (a)

$$a = \frac{D_P}{D_T}$$

- When $a < 1$, the propagation delay is less than the transmission delay. In this case, the frame is sufficiently long that the first bits of the frame will arrive at the destination before the source has completed the transmission of the frame.
- When $a > 1$, the propagation delay is greater than the transmission delay. In this case, the sender completes transmission of the entire frame before the leading bits of that frame arrive at the receiver.
- Note that for $a > 1$, the line is always underused and even for $a < 1$, the line is inefficiently used. However, there are factors other than a affecting utilization, like link protocol
- In essence, for very high data rates, for very long distances between sender and receiver, stop-and-wait flow control provides inefficient line utilization.

Link Utilization

- Since the link can never be completely used all the time, link utilization (u) can be calculated based on link measure (a)

$$u = \frac{1}{1 + 2a_T}$$

Have you ever been to Tim Horton's drive through?

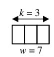
SLIDING WINDOWS

Introduction to Sliding Window Protocols

1. **Piggybacking** is about loading outgoing data frames with control information.
   - + Typical piggybacking piles up multiple ACKs
   - + Control information gets free ride over data frames → better BW utiliz.
   - ? How long should DLL wait for the piggybacked ACKs?
   - ∴ Combined mechanisms of piggyback wait and timeouts can work.
2. Sliding window protocols define the max number of frames that can be sent/received in a sequence without actively waiting for (initiating) an acknowledgement.
   - The sliding window is defined by n, the number of bits required to carry the frame sequence number.
   - At any point of time the sender sends frames only if there is a room in the *(sending window)*. After that sending is paused, and the DLL shuts off NW Layer.
   - At any point of time the receiver accepts frames only if there is a room in the *(receiving window)*. After that arriving frames are discarded.
   - Sliding window does not change the order in which packets arrived or delivered.

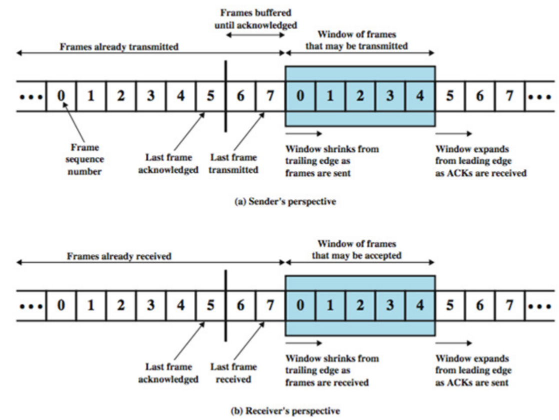
Sliding Windows Flow Control

- allows multiple numbered frames to be in transit
  - receiver has buffer W long
  - transmitter sends up to W frames without ACK
  - ACK includes number of next frame expected
  - sequence number is bounded by size of field (k)
    - frames are numbered modulo 2^k
    - giving max window size of up to $2^k - 1$

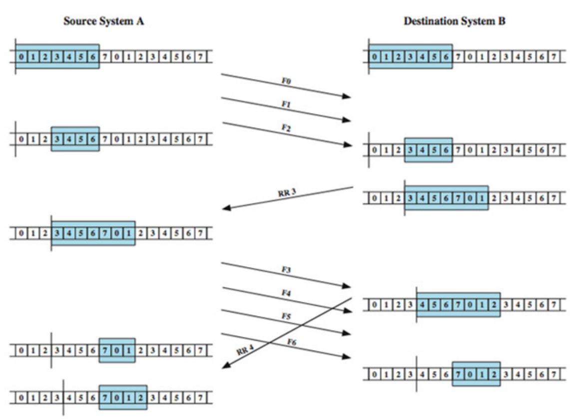


  - receiver can ACK frames without permitting further transmission (Receive Not Ready)
  - must send a normal acknowledge to resume
- if have full-duplex link, can piggyback ACKs

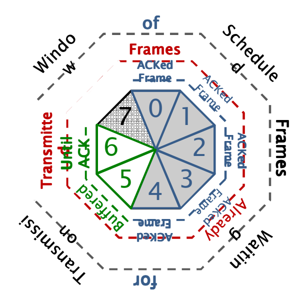
Sliding Window Diagram



Sliding Window Example



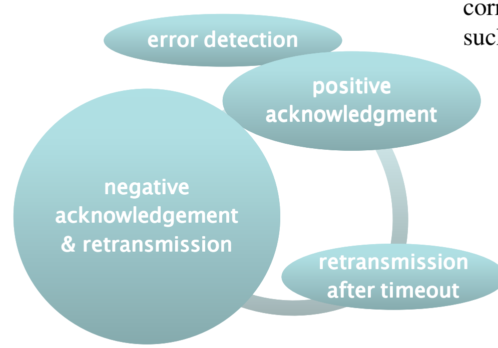
Sliding Window



How to Examine a Protocol

- There are three possible frame transmission outcomes:
  - Correct frame
  - Lost frame
  - Grabbled frame
- There are three possible control frame (ACK/NACK/REJ) transmission outcomes:
  - Correct control frame
  - Lost control frame
  - Grabbled control frame
- Protocols must be examined to ensure its handling of each of the above possibilities
- Protocols must be examined to ensure its handling of each possible combinations of the above possibilities
- Protocol must be able to recover from any possible scenario and never get stuck in infinite loop

Error Control Techniques



detection and correction of errors such as:

lost frames
-a frame fails to arrive at the other side

damaged frames
-frame arrives but some of the bits are in error

Automatic Repeat Request (ARQ)

- collective name for error control mechanisms
- effect of ARQ is to turn an unreliable data link into a reliable one
- versions of ARQ are:
  - stop-and-wait
  - go-back-N
  - selective-reject

Stop and Wait ARQ

- source transmits single frame
- waits for ACK
  - no other data can be sent until destination's ACK arrives
- if receiver get a damaged frame, discard it
  - transmitter will timeout
  - if no ACK received within certain timeout, retransmit
- if ACK is damaged, transmitter will not recognize it
  - transmitter will retransmit after timeout
  - receiver gets two copies of frame
  - use alternate numbering and ACK0 / ACK1

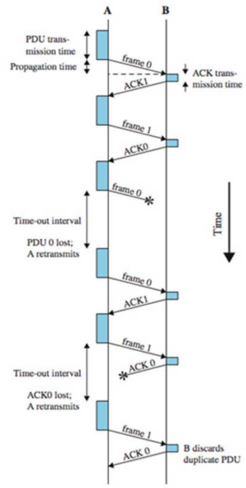
Stop and Wait ARQ

pros

- simplistic

cons

- inefficient



Go-Back-N ARQ

- most commonly used error control
- based on sliding-window
- use window size to control number of outstanding frames
- if no error, ACK as usual
- if error, reply with rejection
  - destination will discard that frame and all future frames until frame in error is received correctly
  - transmitter must go back and retransmit that frame <u>and all subsequent</u> frames

Go Back N - Handling

- Damaged frame
  - error in frame ***i*** so receiver rejects frame *i*
  - transmitter retransmits frames from ***i***
- Lost frame
  - frame ***i*** lost and either
    - transmitter sends ***i*+1** and receiver gets frame ***i*+1** out of sequence and rejects frame ***i***
    - or transmitter times out and sends ACK with ***P*** bit set which receiver responds to with ACK ***i***
  - transmitter then retransmits frames <u>from</u> ***i***

Go Back N - Handling

Damaged Acknowledgement

- **receiver gets frame i, sends ACK (*i*+1) which is lost**
- **ACKs are cumulative, so next ACK (*i*+*n*) may arrive before transmitter times out on frame i**
- **if transmitter times out, it sends ACK with P bit set**
- **can be repeated a number of times before a reset procedure is initiated**

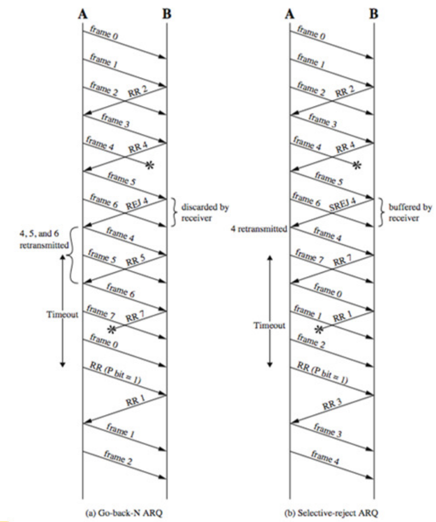
Damaged Rejection

- **reject for damaged frame is lost**
- **handled as lost frame when transmitter times out**

Selective-Reject (SREJ-ARQ)

- also called selective retransmission
- only rejected frames are retransmitted
- subsequent frames are accepted by the receiver and buffered
- minimizes retransmission
- receiver must maintain large enough buffer
- more complex logic in transmitter
  - less widely used
- useful for satellite links with long propagation delays

Go-Back-N vs. Selective Reject



Types of Error

- an error occurs when a bit is altered between transmission and reception
  - binary 1 is transmitted and binary 0 is received or binary 0 is transmitted and binary 1 is received

single bit errors

0 1 0 0 1
0 1 0 1 1

- isolated error that alters one bit but not nearby bits
- caused by white noise

burst errors

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

- contiguous sequence of *B* bits where first and last bits and any number of intermediate bits are received in error
- caused by impulse noise or by fading in wireless
- the effect is greater at higher data rates

Random Error

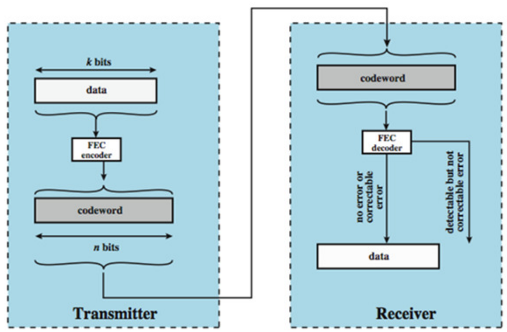
0 1 0 0 0 1 1
0 0 0 1 0 0 1

- Caused by interference in wireless

Error Detection

- regardless of design you will have errors
- can detect errors by using an error-detecting code added by the transmitter
  - code is also referred to as ***check bits***
- recalculated and checked by receiver
- still chance of undetected error
- parity
  - parity bit set so character has even (even parity) or odd (odd parity) number of ones
  - even number of bit errors goes undetected

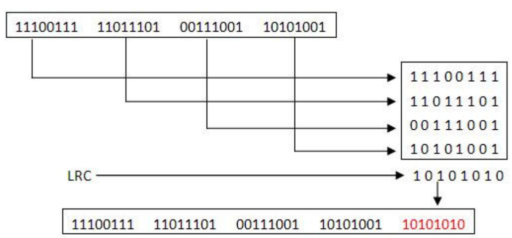
Error Detection Process



How can we detect errors using (odd/even) parity

PARITY CHECK

Longitudinal Redundancy Check



Original data message including LRC

In longitudinal redundancy method, a BLOCK of bits are arranged in a table format (in rows and columns) and we calculate the parity bit for each column separately. The set of these parity bits are also sent along with our original data bits.

Longitudinal redundancy check is a bit by bit parity computation, as we calculate the parity of each column individually.

This method can easily detect burst errors and single bit errors and it fails to detect the 2 bit errors occurred in same vertical slice.

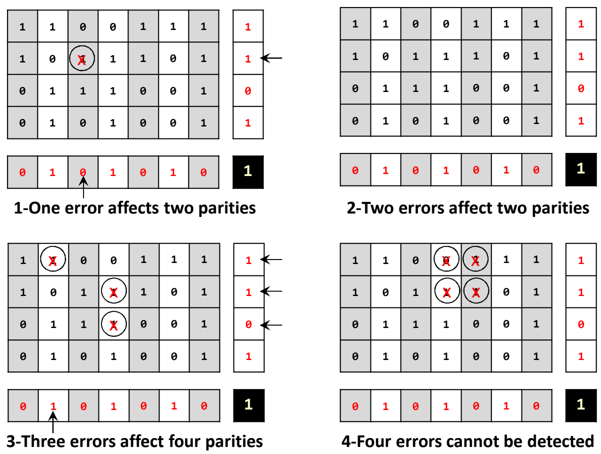
Two Dimensional Parity Check

							Row (even) Parities
1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

Column (even) Parities

Simple Design of row and column parities

Error Detection with Parity Check



Pros & Cons of Simple Parity

- Simple, fast, easy to implement
- Can determine if the message has errors, but may not detect the locations of the error bits.
- Even errors in happening in the same column/row will go undetected

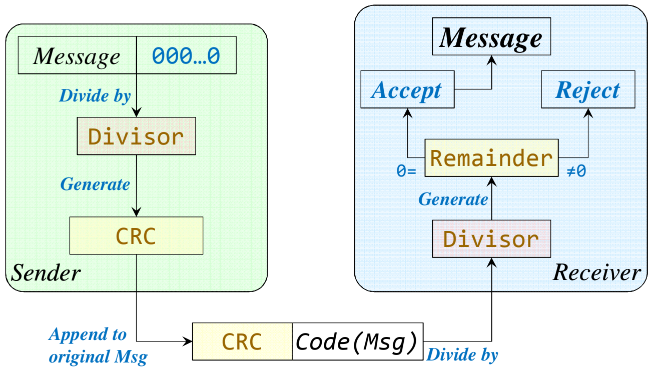
How can we detect errors

CYCLIC REDUNDANCY CHECK (CRC)

Cyclic Redundancy Check (CRC)

Example: The source encodes the 4-bit original frame **m**=(m_3, m_2, m_1, m_0) into a 7-bit code word **x=** (x_6, x_5, x_4, x_3, x_2, x_1, x_0) and then sends the coded word into the network. The receiver receives a coded word, that is, potentially, corrupted. The goal for the receiver is to detect and correct the possible error in transit, and to restore the original frame (m_3, m_2, m_1, m_0) using CRC algorithm.

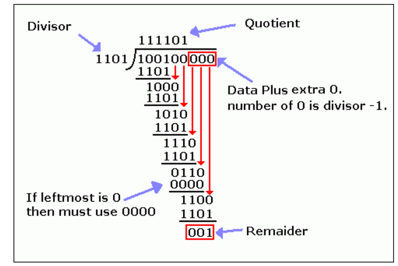
The CRC Process



 41

Generating CRC 1

A CRC generator uses modulo-2 division. In the first step, the four-bit divisor is subtracted from the first four bits of the dividend. Each bit of the divisor is subtracted from the corresponding bit of the dividend without disturbing the next higher bit.



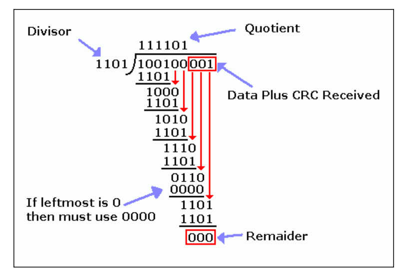
 42

Generating CRC 2

In the process, the divisor always begins with a 1; divisor is subtracted from a portion of the previous dividend/remainder that is equal to it length; the divisor can only be subtracted from a dividend/remainder whose leftmost bit is 1. Anytime the leftmost bit of the dividend/remainders is 0, a string of 0s, of the same length as the divisor, replaces the divisor in the step process. This restriction means that, at any step, the leftmost subtracted will be either 0-0 or 1-1, both of which equal 0. So, after subtracted, the leftmost bit of the remainder will always be a leading zero, which is drop off, and the next unused bit of the divided is pulled down to fill out the remainder. Note that only the first bit of the remainder is drop, if the second bit is 0, it is retained, and the dividend/remainder for the next step will begin with 0. This process repeats until the entire dividend has been used.

 43

Checking CRC



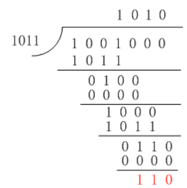
A CRC checker functions exactly like the generator. After receiving the data appended with the CRC, it does the same module-2 division. If the remainder is all 0s, the CRC is dropped and the data is accepted; otherwise, the received stream of bits is discarded and data are resent.

 44

CRC: Frame Encoding

The CRC construction can be divided into two parts: encode and decode.

Encode: First, the frame (1001) is extended to 7-bit (1001000) by attaching three 0s. Second, let the 7-bit (1001000) divide a special ***divisor*** (1011) and get the remainder (110) (see division). Third, substitute the attachment of three 0s by the remainder, so ones transform (1001000) into another 7-bit (1001110). This is the code word.



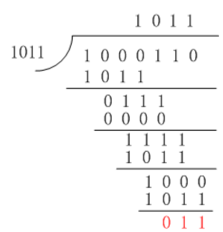
CRC: Frame Decoding (1)

Decode: Assume the error bit is y_3 in this example, so the received 7-bit sequence is **y**= (y_6, y_5, y_4, y_3, y_2, y_1, y_0)=(1000110). The receiver divides this sequence by the same divisor as the source, and obtain the remainder (011). The remainder is normally named as ***syndrome***. Now we come to the key point of error correction. Actually, it is by the syndrome that the receiver can find the error position and correct the error. In other words, there is one-to-one correspondence between the syndrome and the error pattern (see next page).

CRC: Frame Decoding (2)

In this example, the receiver identify the error bit being y_3 by the ***syndrome***=(011) so it reverses the error bit to '1'. At last, it extracts the original frame (1001) from the first 4 bits of the corrected words (1001110).

Can we improve error correction by adding more check bits?



Pros & Cons of CRC

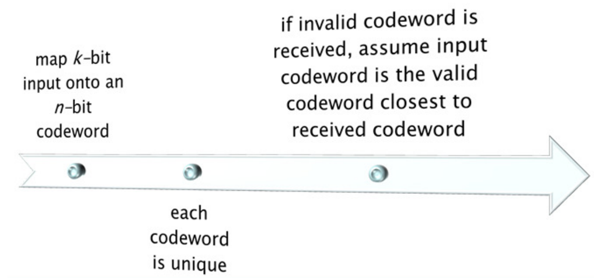
- Any single bit error can be detected and possible corrected.
- Any burst error of length (L<c-1) will be detected
  - L: is the length of the burst error
  - c: is the highest order of the divisor polynomial
- Can determine if the message has errors, but may not detect the locations of the error bits.
- Not reliable for error correction

How can we detect and correct errors

HAMMING CODES

How Error Correction Works

- adds redundancy to transmitted frame
  - redundancy makes it possible to deduce original frame despite some errors
- block error correction code



Hamming Distance

```
10001001
10110001
00111000
```

- An n-bit codeword is a single data symbol (character)
- Hamming distance = # of differences bet. codewords.
- For codeword `1011 010_` → Even parity `1011 0100`,
  → odd parity `1011 0101`.

- To **detect** e errors, you need $d=e+1$ code
- To **correct** e errors you need $c=2e+1$ code

```
a:00000 00000
b:00000 11111
c:11111 00000
d:11111 11111
```

- Check this error-correction example:
  - Consider a code with only 4 valid code words.
  - This code has a distance=5, (i.e. can fix 2 errors; $e=2$) if you receive `00000 00111`, you know it must've been `b`.
  - If 3 errors: `00000 00000` becomes `00001 10011`, we cannot fix it (`a`/`b`).

Hamming Codes

- Bits of the codeword are numbered: bit 1, bit 2, ..., bit n.
- Check bits are inserted at positions 1,2,4,8,.. (all powers of 2). The rest are the m data bits.
- Rewrite original frame after inserting empty spaces for parity.
  - In other words; this codeword `1001 1010`; becomes this codeword `_ _ 1 _ 0 0 1 _ 1 0 1 0`.
- Now the problem is how to calculate the Hamming parity bits? (the underlined spaces)

Calculating the Hamming Code

- The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:
  1) Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
  2) All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
  3) Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
     Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)
     Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3, 6,7, 10,11, 14,15, ...)
     Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4-7, 12-15, 20-23, ...)
     Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15, 24-31, 40-47, ...)
     Position 16: check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc. (16-31, 48-63, 80-95, ...)
     Position 32: check 32 bits, skip 32 bits, check 32 bits, skip 32 bits, etc. (32-63, 96-127, 160-191,...)
  4) Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Hamming Code Example

- For this byte of data **1001 1010**, create the code word using even parity.
- Lets rewrite the data leaving spaces for the parity bits
  _ _ 1 _ 0 0 1 _ 1 0 1 0.
- **Position 1**
  checks bits 1,3,5,7,9,11: Even so set position 1 parity to 0
- **Position 2**
  checks bits 2,3,6,7,10,11: Odd so set position 2 parity to 1
- **Position 4**
  checks bits 4,5,6,7,12: Odd so set position 4 parity to a 1
- **Position 8**
  checks bits 8,9,10,11,12: Even set position 8 parity to 0

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12
Original Msg	1	0	0	1	1	0	1	0				
New Msg	0	1	1	1	0	0	1	0	1	0	1	0

Finding the Error in Hamming Code

- Suppose the word was originally **100** then it was coded as **111000** but was received as **111001** instead due to a white error (even parity).
- How can the receiver calculate which bit was wrong and correct it? Receiver assumes data was **101**
  - Check bit #1: 1 - checks bits 3,5 - 1 0 - OK
  - Check bit #2: 1 - checks bits 3,6 - 1 1 - WRONG
  - Check bit #4: 0 - checks bits 5,6 - 0 1 - WRONG
- The bad bit is bit 2 + 4 = bit #6.
  Data was corrupted. Data should be 100.

Comments on Hamming Codes

- Hamming codes cannot discover errors happening in the parity bits.
- Hamming codes can only fix single error.
- To fix burst errors, transmission takes place on a column bases rather than codeword.

```
00110010000
10111001001
11101010101
11101010101
01101011001
01101010110
01111001111
10011000000
11111000011
10101011111
11111001100
00111000101
```

Order of bit transmission

Error-Detecting Codes

- A **burst error** doesn't mean all bits are wrong, it means at least first and last are wrong, the middle once could be wrong.
- Each ready block can be viewed as a rectangular matrix of n bits wide and k bits height. One parity bit is appended at the end of each column composing one new row.

$$\begin{array}{c|l} \multicolumn{1}{c}{n} & \\ 10011000000 & \\ 11111000011 & \\ 10101011111 & k \\ 11111001100 & \\ 00111000101 & \end{array}$$

- Receiver validates arriving data against column-wise parity. Retransmission takes place until block arrive correctly.
- This detects single burst of length n since only one bit per column will be changed. A burst of $n+1$ will go undetected.

Pros & Cons of Hamming Codes

- Any single bit error can be detected and corrected.
- Any burst error of length ($L<3$) will be detected and corrected
  - L: is the length of the burst error
- Reliable with its own limitation

Error Detection vs. Error Correction

- Detection
  - Pro: Overhead only on messages with errors
  - Con: Cost in bandwidth and latency for retransmissions
- Correction
  - Pro: Quick recovery
  - Con: Overhead on all messages
- What should we use?
  - Correction if retransmission is too expensive
  - Correction if probability of errors is high
  - Detection when retransmission is easy and probability of errors is low

Now we quickly review main highlights of the ISO/OSI HDLC

HIGH-LEVEL DATA LINK CONTROL (HDLC)

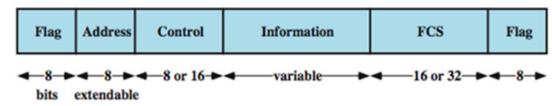
High-Level Data Link Control (HDLC)

most important data link control protocol
- specified as ISO 3009, and ISO 4335
- basis for other data link control protocols

station types:
- Primary - controls operation of link
- Secondary - under control of primary station
- Combined - issues commands and responses

link configurations
- Unbalanced - 1 primary, multiple secondary
- Balanced - 2 combined stations

HDLC Frame Structure



(a) Frame format

– uses synchronous transmission
– transmissions are in the form of frames
– single frame format used

Flag Fields and Bit Stuffing

- delimit frame at both ends with 0111 1110
- receiver hunts for flag sequence to synchronize
- bit stuffing used to avoid confusion with data containing flag sequence 0111 1110
  - 0 inserted after every sequence of five 1s
  - if receiver detects five 1s it checks next bit
  - if next bit is 0, it is deleted (was stuffed bit)
  - if next bit is 1 and seventh bit is 0, accepted as flag
  - if sixth and seventh bits 1, sender is indicating abort

Original Pattern:

111111111111011111101111110

After bit-stuffing

1111101111101101111101011111010

Bit Stuffing

- The flag byte = [0111-1110]
- Any sequence of than 5 ones [0111-1101]
  is stuffed to be [0111-1100] [1...

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

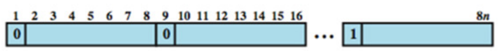
Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing
(a) The original data.
(b) The data as they appear on the line.
(c) The data as they are stored in receiver's memory after destuffing.

Address Field

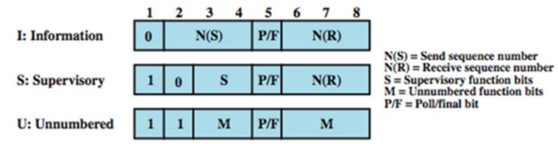
- identifies secondary station that transmitted or will receive frame
- usually 8 bits long
- may be extended to multiples of 7 bits
  - leftmost bit indicates if is the last octet (1) or not (0)
- address 11111111 allows primary to broadcast



(b) Extended Address Field

 65

Control Field



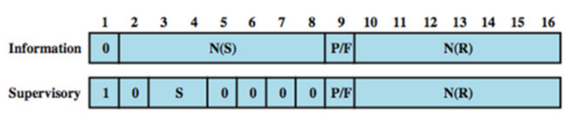
(c) 8-bit control field format

- different frame types
  - Information - data transmitted to user (next layer up)
    - flow and error control piggybacked on information frames
  - Supervisory - ARQ when piggyback is not used
  - Unnumbered - supplementary link control functions
- first 1-2 bits of control field identify frame type

 66

Control Field

- use of Poll/Final (P/F) bit depends on context
- in command frame P bit set to 1 to solicit (poll) response from peer
- in response frame F bit set to 1 to indicate response to soliciting command
- sequence number usually 3 bits
  - can extend to 8 bits as shown below



(d) 16-bit control field format

 67

Summary

- data link protocols
  - frame synchronization
  - flow control
  - ARQ
    - stop-and-wait,
    - Go-Back-N
    - Selective Reject
- Common Algorithms
  - sliding window
  - ACK/NCK frames
  - Timeouts

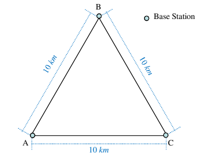
 68

Summary

- error detection & correction
  - Hamming
  - CRC
- HDLC
  - Bit Stuffing
  - Address fields

 69

Example



Consider a microwave communication link with three stations (A, B, C) equally spaced from each other as in the diagram shown. The microwave links provide a data rate of 10 Mbps over a length of 10 km.

i. What is the mean time to send a frame of 1000 bits from any station to another? Consider measuring from the beginning of transmission to the end of reception; and assume a propagation speed of 200 m/μs.

ii. If stations (A & C) begin to transmit at exactly the same time targeting station B, their packets will interfere with each other. If each transmitting station monitors the link during transmission; how long (in seconds and in bits) before the sending station notices the interference?:

 70

Solution

i) The distance between stations is 1.0 km. The time to send any frame equals the transmission delay plus propagation delay.

D_T is the transmission delay:

$$D_T = \frac{No.of\ bits\ in\ frame}{bit\ rate\ of\ the\ transmission\ link\ (bps)} = \frac{1000}{10\times10^6} = 10^{-4} sec = 100\ \mu sec$$

D_P is the propagation delay:

$$D_P = \frac{Distance\ between\ nodes\ (m)}{speed\ of\ electromag\ wave(\frac{m}{\mu sec})} = \frac{10\times1000}{200} = 50\mu sec$$

Total time consumed for transmission and propagation = 100 + 50 = 150 μsec

ii) The time to sense the interference = the time required to propagate the first bit

(you may add the time to transmit the first bit, but that would be negligible.
Therefore, the time before you sense interference = 50 μsec

Then, the question now is how many bits will be transmitted in 50 μsec.

Since $D_T = \frac{No.of\ bits\ in\ frame}{bit\ rate\ of\ the\ transmission\ link\ (bps)} = \frac{F}{10\times10^6} = 50\times10^{-6}\ sec$

$F = 500\ bits$

 71

Review this part if you are missing on math

APPENDIX ON:
MODULO 2 ARITHMETIC

 72

Modulo 2 Arithmetic

- Addition Rules

0 + 0 = 0	Ex:	1011
0 + 1 = 1		+110
1 + 0 = 1		1101
1 + 1 = 0		

Modulo 2 Arithmetic

- Division

Ex: **1011 | 1001011**

^ ^

divisor dividend

Modulo 2 Arithmetic

- Division

1

Ex: **1011 | 1001011**

1011

Modulo 2 Arithmetic

- Division

1

Ex: **1011 | 1001011**

1011

0010

Modulo 2 Arithmetic

- Division

```
              10
Ex:   1011 | 1001011
             1011
             00100
```


Modulo 2 Arithmetic

- Division

```
              101
Ex:   1011 | 1001011
             1011
             001001
```


Modulo 2 Arithmetic

- Division

```
              101
Ex:   1011 | 1001011
             1011
             001001
               1011
```


Modulo 2 Arithmetic

- Division

```
              101
Ex:   1011 | 1001011
             1011
             001001
               1011
               0010
```


Modulo 2 Arithmetic

- Division

```
            1010
Ex:   1011 | 1001011
             1011
             001001
               1011
               00101
```


Modulo 2 Arithmetic

- Division

```
            1010        ← Quotient
Ex:   1011 | 1001011
             1011
             001001
               1011
               00101    ← Remainder
```