In [1]:

```
#######################
# Project 3
# ESE 572
# Layer 2 Channel Coding
#######################

# Choose a probability of bit error to use when simulating the receiving end.

prob = 0.1
# prob = 0.01
```

In [2]:

```
#######################
# Step 1
#######################

# Function to convert string to binary using ASCII encoding
def string_to_binary(string):
    binary = ''
    binaryS = ''
    binaryC = ''
    counter = 0
    for char in string:
        ascii_code = ord(char) # Get ASCII code of character
        if counter == 24:
            binary += format(ascii_code, '08b')
            binaryS += format(ascii_code, '08b') + ' '
            binaryC += format(ascii_code, '08b') + '\n'
            counter = 0
        else:
            binary += format(ascii_code, '08b')
            binaryS += format(ascii_code, '08b') + ' '
            binaryC += format(ascii_code, '08b')
            counter += 1
    return binary, binaryS, binaryC

# Load the document
filename = 'input.txt'
text = ''
with open(filename,'r') as i:
    text += i.readline()

binary_data, binaryS_data, binaryC_data = string_to_binary(text)

# binary_data is single line
# binaryS_data contain spaces to delimitate each character
# binaryC_data is split as chunks of 200 bits
```

In [3]:

```
#######################
# Step 2
#######################

# Create CRC for g(D) = [D16 + D12 + D5 + 1]
crc_str = '10001000000100001' # divisor
int_crc_str = int(crc_str,2)
frames = []

binary_chunks = binaryC_data.split('\n')
for chunk in binary_chunks[:-1]:
    temp = chunk + "0000000000000000"
    temp = int(temp,2)
    crc = temp % int_crc_str
    crc = format(crc, '016b')
    if len(crc) > 16:
        crc = crc[-16:]
    frames.append(chunk + crc)

print((frames[0]))
```

```
010010010110111000100000001100010011100000011010100110011001011000010000000111000001110010011011110110110101101001011011100
1100101011011100111010000100000010100110111010000101110001000000010011000110111111111100010111010
```

In [4]:

```
#######################
# Step 3
#######################
## Channel Coding **NEW**

import numpy as np

def encode_message(msg, matrix):
    code = np.mod(np.dot(msg, matrix), 2)
    return code
```

```python
def decode_message(code, valid_codewords):
    incorrect_vec = []
    incorrect = 0
    for codeword in valid_codewords:
        for ind in range(len(codeword)):
            if codeword[ind] != code[ind]:
                incorrect += 1
        incorrect_vec.append(incorrect)
        incorrect = 0
    closest = np.min(incorrect_vec)
    closest = np.where(incorrect_vec == closest)[0][0]
    decoded_msg = valid_data[closest]
    return decoded_msg

G = np.array([[1,1,1,1,1,1,1,1],
              [0,0,0,0,1,1,1,1],
              [0,0,1,1,0,0,1,1],
              [0,1,0,1,0,1,0,1]])

valid_data = [[0,0,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,1],
              [0,1,0,0], [0,1,0,1], [0,1,1,0], [0,1,1,1],
              [1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1],
              [1,1,0,0], [1,1,0,1], [1,1,1,0], [1,1,1,1]]

valid_codewords = []
for data in valid_data:
    valid_codewords.append(encode_message(data,G))
print(valid_codewords)

############ Example ############
msg = np.array([0, 1, 1, 1])
print("Sent Message:", msg)
print()

code = encode_message(msg, G)
print("Encoded code:", code)
print()

decoded_msg = decode_message(code, valid_codewords)
print("Decoded message:", decoded_msg)
############ End Example ############

def array_to_string(array):
    output = ""
    for each in array:
        output += str(each)
    return output

def string_to_array(string):
    output = []
    for each in string:
        output.append(int(each))
    output = np.array(output)
    return output

def RM_string(input_string):
    output_string = ""
    ind = 0
    while(ind <= len(input_string)):
        dat = input_string[ind:ind+4]
        data_string = string_to_array(dat)
        if data_string.size != 0:
            data_string = data_string.reshape((1,4))
            encoded = encode_message(data_string,G)[0]
            output_string += array_to_string(encoded)
        else:
            output_string += str(dat[-1:])
        ind += 4
    return output_string

binaryRM = []
for frame in frames:
    binaryRM.append(RM_string(frame))

print(len(binaryRM[0]))
```

```
[array([0, 0, 0, 0, 0, 0, 0, 0]), array([0, 1, 0, 1, 0, 1, 0, 1]), array([0, 0, 1, 1, 0, 0, 1, 1]), array([0, 1, 1, 0, 0,
1, 1, 0]), array([0, 0, 0, 0, 1, 1, 1, 1]), array([0, 1, 0, 1, 1, 0, 1, 0]), array([0, 0, 1, 1, 1, 1, 0, 0]), array([0,
1, 1, 0, 1, 0, 0, 1]), array([1, 1, 1, 1, 1, 1, 1, 1]), array([1, 0, 1, 0, 1, 0, 1, 0]), array([1, 1, 0, 0, 1, 1, 0, 0]),
array([1, 0, 0, 1, 1, 0, 0, 1]), array([1, 1, 1, 1, 0, 0, 0, 0]), array([1, 0, 1, 0, 0, 1, 0, 1]), array([1, 1, 0, 0, 0,
0, 1, 1]), array([1, 0, 0, 1, 0, 1, 1, 0])]
Sent Message: [0 1 1 1]

Encoded code: [0 1 1 0 1 0 0 1]
```

```
Decoded message: [0, 1, 1, 1]
432
```

In [5]:

```
########################
# Step 4
########################

# flag = '01111110'
flag = '000111111111111111111000' # tripled
framed_frames = []
bitstuff=0
for binRM in binaryRM:
    i = 0

    while(i <= (len(binRM) - 18)):
        if binRM[i:i+18] == '111111111111111111':
            binRM = binRM[:i+17] + '0' + binRM[i+17:] # bit stuff after any string of five 1s
            bitstuff+=1
            i += 18
        else:
            i += 1
    framed_frames.append(flag + binRM)

framed_frames.append(flag) # after last FEC frame, insert another flag
print('bits stuffed: '+str(bitstuff))
print(framed_frames[0])
```

```
bits stuffed: 0
000111111111111111111110000000111110101010001111001100001100110011000000000110011001010101011001101111111101100110010110100
110011001100110001100111111100000011001100000000001101001000000000110100100110011001111001001011000111100101001010011110010
1010100011110011000011001111000101101000111100110000110110100100001111001100110000000001011010011001100110100100001111001
10011111000011001100110000000000000111111110000001111001001011010010110111111111100110011100 1100
```

In [6]:

```
########################
# Sequence to Transmit
########################

sequence = ''
for fram in framed_frames:
    sequence += fram
    # print(len(fram))

print(len(sequence))
```

```
20544
```

In [7]:

```
########################
# Step 5: Error
########################

import random

lim = {0.1: 9,
       0.01: 99}

rxbits = ''
errors = 0

for bit in sequence:
    rnum = random.randint(0,lim[prob])
    # rnum = 1 # no errors
    if rnum == 0: # error bit
        errors += 1
        if bit == '1':
            bit = '0'
        else:
            bit = '1'
    rxbits += bit
print("Number added random errors:", errors)
print(len(rxbits))
```

```
Number added random errors: 2077
20544
```

In [8]:

```
########################
# Step 5: Unstuffing
########################

def unstuff(bits):
    i = 0
    while i < len(bits)-3:
        #print(bits[i:i+19])
        if bits[i:i+19] == '1111111111111111101':
            print(bits[i:i+19])
            bits = bits[:i+17] + bits[i+18:] # remove bit stuffing
            print(bits[i:i+19])
            i += 18
        else:
            i += 1
    return bits

rxbits = unstuff(rxbits)
print(len(rxbits))
```

```
20544
```

In [9]:

```
########################
# Step 5: Fixing
########################

corrected_bits = rxbits
# finding flags to reseparate frames
i = 0
start = -1
end = -1
new_frames = []
flag_flag = 0

while (i < len(corrected_bits)):
    if corrected_bits[i:i+24] == flag:
        flag_flag = 1
        if start != -1:
            end = i
            temp = corrected_bits[start:end]
            new_frames.append(temp)
            start = i
        else:
            start = i
    if flag_flag:
        i += 24
        flag_flag = 0
    else:
        i += 1

new_frames.append(corrected_bits[start:])

new_frames = new_frames[:-1]
print(new_frames[0])
print(len(new_frames[-2]))

# frames separated, unstuffed but need :: untripled, CRC checked, and then converted to text
```

```
0001111111111111111111100010001111101010100011110011000011001100110000000000010011001010100001001101111001101100110110110110
1100100011001100011001110110011101110110000000000110100100000000001100001001100111011110101010110000111001010010100111111010
1011100011110001000011001111000101101001001100110000110111100000000111000110011001000000101101001100111011011010000101100 1
1001111100101001100011000000000001111111110000011110010010101100100001111110110011101110011000001111011111011110110000110
0001010010100011110010101110011010110110111100110011000000000001111001010010110111110010111100110101100111011001011000100 1
1100011110011111111001110010101010000111100110000110110100101001111001100111000000101101101001101001000111000101010101001 0
0110001010001111001010010100111100010101010011100010000011001100110000000000000111101101110111011010111001001111000100111 1
0010010011110100100111111000000000001111101010010111110010010110100001110111111111101111000000111100010111011011110011000011
0011110000000110001100110000000000011100111111110011110101010100110110101010100001100100011000101001000000001000001111000
1010101011000110111011001101001000011010011110010010110111010010011001100110011111100000010101101100000011110100110100100
111100001010100010100001110001001111001110001000101100101010000011111001010001001111001010011101110010000000010000111101 1
01001010110101100000101010001100000000
456
```

In [10]:

```
########################
# Step 5: Remove flag
########################

rxseqs = []
for nf in new_frames:
    rxseqs.append(nf[24:])

print(len(rxseqs[0]))
# need :: CRC checked, and then converted to text
```

```
1344
```

In [11]:

```
########################
# Step 5: Decode RM
########################

temp_rxseqs = []
for frame in rxseqs:
    # frame = frame[:-8]
    temp_frame = ""
    count = 0
    while(count < len(frame)):
        message = string_to_array(frame[count:count+8])
        decoded_msg = decode_message(message, valid_codewords)
        temp_frame += array_to_string(decoded_msg)
        count += 8
    temp_frame += frame[count:]
    temp_rxseqs.append(temp_frame)
    print(temp_frame)

rxseqs = temp_rxseqs
# print(rxseqs)
print(len(rxseqs[0]))
```

```
010010010110111000100000001100010011001000110101001100110010001000100000001110000001110010011000010110110101101001011011100
1100101101011100101010000100000010100110111010000101101001000000100110001100001000010001011101001001000010101110101011010
0101110011001000000110110101100101011100100110001101101000001000010110111001110100001000000101011101100001011110010110110
10110000101101110001000000010000110111001001011111011101110010000001101101110001010001100001010110000101101110011000000010
00000110100001101001011100010010001000000000001100001001000110111010001101111011100100010110000100000010101110110100100000
00101100011011010010110000101101101001000000010001110101111000010000
01110010011000010110010101101110100001110011010101100001011001100010000000100010101101100011010010110111101110100001000000
10010100010000001001000100101100001000000110001101101111011011100110001101100101100100000110010101001000110001110010001011
1001100101011001000010000000110000101100010011011110001011001110100001000001000010000000100001100101001000000011011000110000
101100011011000000001000000011011111110001100010000000110100101101110000000010000100100001100001100011001101110100011010010111
010001110101011101000000110010011111101101110011100110010000001101111011001100010000000000100000011001010001110110100001100
1010001001000100000011011000110010101100001000100100001111100010011010010001100011011100110100101101110011001110001000001
100100011011100010000000110100011010000001101010010000000110011101110010011011010111011110101001011011100110011100100000011
0110101101001011001000111011101100101000110011010000101000111110001010011011101000010110000100000011011000110010101100100
0010000000111000101101000011000100000000000110001001101111011101010110111001100100011010010110111001100111001000000110111110
1100110001000000000000000011001111010010101001000010110100001011100110001100001100000011011100110011101110100011011110111011
1000100000001010101011000100010101001110110011000100101001001110011011010010111010001111001001000000011000000011001101010100
000000001100000000011101011010010001100011101000010111000100000001001100011011110111010101101001000000011001011100010000000100
0000000000000111001000000010001100010111101110010000000111010001101000000000101001000000001100000011110000001101000011000001110
101110010010010100001010111001100010000001100001011011100110010000100000000000101001100000000000110010110000100000001110111 01
1000010111011000100000001110011001000000011011110110000000010000000001001011011010110000101101001011001110000111000000 0111100
0100011000111001001100001011001000111010001110011001000010110011001100001000111110110111101100100011001010110010000100000
011010010110111001110100011011110010000000101001101110100001001000111000000100110001101111000110101010011001001000011101110
101011000100111001100011100001000000011000100110111101101111010000110111010001101001011000010110011100100000001110100011010
000110010100100000000000000011011110111000001110000001100000001100001001101000000111100110100101001000110001101001011011110 11
011100010000000001111000000110001000000011101000110100001100101001000000001101010110111101110101011000000011001110010000 00110
0011010010010111010000101001001011100010000000101011101101001000000011000000110010010001001011101000110100000010000000111 01000
1100100000000101011100110110111000100000001101110011001010111011101100011011001010110001101101011001100100001001100100000 101
100011011000010110001001100101001000000011000010101100001101110
001000000010000000011100100010001001110011011100110110100101101110011001110010000000110111001100101011001010010010000100000 00
110000101101111011100100010000000110010101100100000000101011000110110000101110000110111000101 0111
011010010110111101101110001000000010110100100000000100010011011110111010001101000000100000001101001011011100110010001110 0100
0010011011101000010000001101001010000001011011000010000000001010001110010011000011111110000001010000001100011000110001001 1011
1001101001011011100100011100100000001100011011011100110010000100000001100010011000010111001101101001011000110010000000110 011
1011001010110111001100101001000100110000101101100001000000011000110000101010110100000011000110001101111011101010101001001 11
0011011001010111000000010000000010001000100000000000011011000100110111001100100011101010110001101110100011001010110010000 100
0000110111101110101011101000111001101100011011001000000000110100010001000101110001100101001000000101101010011011000100000 01
1011100001111101110010011011010110000101101100001000000011101110110111101110010000010110110100100101110011001110000000001 1
0100001101111011101010100001000110011110100000001011110100100001010010111000100000000100011011011110010000000111010001011 000
0100010100100000001100110011010010111001001110011001111100010000000110010101100100011101010110000100100001011101000110101 11
0101111011011100110000101100100100010010001010011000110110000100000001110011011101000010010101110000000100000011011110110 01
1000100000001110100011010000110010100100000001111001000000010011101010000111001100111001000000000001110110000101110011011 0100
0011000111100010001101001010001010110011011100110011101110100011010110110111000100000001010100011011100110100101110000011 0
0001011100101101001001101001011101000111100100100000001110111011000010111001100100000001110100011011110010000001100101111 00
0110011100001101000110001110011011101000110000100010010010011000110100101110011011010000001000000010000101100001001000000 01
0001010111011001100101011000110110000000110111001100101001000000000110110111001001101111011001110111001000100101000111111 001
0100011000110000101100000000100000011011110110111000100000010011010110001101110001101111110110001001100101011100100010000 0
0011001000110101010111000010000000011000100111000000110101001101000010111000010000010011111100111000100110010000010101000 10
1100110111001110010001000000011101000110100001100101001000000011100110111010101100011011000110110010101100101011001000110 10
0101101110011000100010000000110010001100101011000110110000101100100011001000011011100111010010001101100011100110010001010 1
1000000111000000110100001100101001000000011000110011111101101110011101000110100101101110101101010110100101100001011000010 000
000000110010001100100011101010110001101100001000001000110100110001100011110000
672
```

In [12]:

```
########################
# Step 5: Check CRC
########################

# Create CRC for g(D) = [D16 + D12 + D5 + 1]
crc_str = '10001000000100001' # divisor
int_crc_str = int(crc_str,2)
failed = 0
total = 0

for rxs in rxseqs:
    crc_rx = rxs[-16:]
    beginning = rxs[:-16] + "0000000000000000"
    beginning = int(beginning,2)
    crc = beginning % int_crc_str
    crc = format(crc,'016b')
    if len(crc) > 16:
        crc = crc[-16:]
    if crc != crc_rx:
        print("crc: "+str(crc)+" crc_rx: "+str(crc_rx))
        failed += 1
    total += 1
    print('Number of Failed Frames with Error Probability ' + str(prob) + ': --- ' + str(failed) + ' --- \n')
```

```
crc: 1110100100000111 crc_rx: 0101111000010000
crc: 0111011110100110 crc_rx: 0101100001101110
crc: 0010001101010010 crc_rx: 1101110001010111
crc: 0100110110011000 crc_rx: 1000110001110000
Number of Failed Frames with Error Probability 0.1: --- 4 ---

Failed Frames: 100.0% of 4 total frames recieved.
```

In [13]:

```
########################
# Convert Back to Text
########################

all_rx_bits = ''
char_arr = []
paragraph = ''
i = 0

for rxs in rxseqs:
    if len(rxs[:-16]) < 200:
        continue
    else:
        all_rx_bits += rxs[:-16]

while i < len(all_rx_bits)-7:
    char_arr.append(all_rx_bits[i:i+8])
    i += 8

for cb in char_arr:
    character = chr(int(cb,2))
    if (character!='~'):
        paragraph += character

print(paragraph)
```

```
In 1253" pramine®T St- LaºHWV  2 ÖW&6  çB v  Ö â 7%÷r ÜQ  an` hiq"a#tor, Wi ciam Graen  5af Eliot J   , conce  eHÇ"æVB &ñg
B@  R Æ 6 üb  à!!  3titut ?ns of  Ghe  lea  HÆæ  æq Fâ F  R w&×z  ær Ö  GvQ  |St, led qhbbounding of g¥HZ 1  æwFöâ V"§
f%'6  G   50 ´  t. Loui . @r b÷ th  0x40uÉ(W1 æBS2Â v b 2 ö  Ö  pàx  radts!fa oded into St$pLo ¦HwV'1Â &öô7F  r F  R ÷  Aæ   io
n   the 5ou`g cIt). Wi  H  F  F@W6â æWv6V6³!2 6 &R  "'76  ær æVRB ÷" V@V6  öâ Ò &÷F   æG!7B  Â G& ÁA  bninG cnd basic gene"a
l c
´ Æ÷U'6W "6&æGV7FVB ÷WG66@4D\e µ6 n rmal wor i.ghouB3Ð/HRâ 6ò E  R f  '3â VGV FºöæH  Ll st%p of the y u g  ashcÄiEfæwF¶â Fæ   -
&  G   v 2 Fò ^3   sta Lish !a Evec`ne  rogr% (Æ  öâ Ö7 ö&W" #UÂ   SBá üâd nr the succeedinb decadd7:FÇ2+   R 3öçF  ëV   FGV6 F
```

In [14]:

```
# In 1853, prominent St. Louis merchant Wayman Crow and his pastor, William Greenleaf Eliot Jr., concerned about
# the lack of institutions of higher learning in the growing midwest, led the founding of Washington University
# in St. Louis. During the 1840s and 50s, waves of immigrants flooded into St. Louis, boosting the population of
# the young city. With these newcomers came a pressing need for education - both industrial training and basic
# general courses - conducted outside of normal working hours. So the first educational step of the young
# Washington University was to establish an evening program on October 22, 1854. Over the succeeding decades,
# the continuing education program underwent many changes. The university flourished at its location in downtown
# St. Louis for its first 50 years, growing from an evening program to an institution offering a full slate of
# scientific, liberal arts and classical course offerings. In time, schools of law and fine arts were added. In
# 1891, the school acquired the St. Louis Medical College to form a medical department, which merged with the
# Missouri Medical College in 1899.
```