Part J

We first process each frame to binary and decode it through the decoder from part D with slight modifications

We define the decode() function to iterate through the frame and pick out the id, data and crc checksums, and ensure all other forms are as expected

In [ ]:
```
def decode(frame):
    frame = bin(frame)[2:]
    n_s = (len(frame)//4 + 1)*4
    frame = frame.rjust(n_s, '0')
    pointer = 0
    id = ''
    if frame[pointer] != '0':
        print('Start of frame bit error')
    pointer +=1

    for i in range(11):
        id += frame[pointer]
        pointer +=1

    if frame[pointer] != '1':
        print('SRR bit error')
    pointer +=1

    if frame[pointer] != '1':
        print('IDE bit error')
    pointer +=1

    for i in range(18):
        id += frame[pointer]
        pointer +=1


    if frame[pointer] != '0':
        print('It is not a data frame')
    pointer +=1

    pointer += 2 # reserved bits we dont care about

    num_bytes = int(frame[pointer: pointer+4], 2)
    pointer +=4

    data = frame[pointer: pointer+(num_bytes*8)]
    pointer += (num_bytes*8)

    crc = frame[pointer: pointer+15]
    pointer +=15

    if frame[pointer] != '1':
        print('crc delimiter error')
    pointer +=1

    ack = frame[pointer]
    pointer +=1
    if frame[pointer] != '1':
        print('ack delimiter error')
    pointer +=1

    if frame[pointer: pointer +7] != '1111111':
        print('end of frame error')
    pointer +=7

    return id, data, crc, ack
```

In [ ]:
```
id1, data1, crc1, ack1 = decode(0x527EAAAA0618B3200003FF)
id2, data2, crc2, ack2 = decode(0x527EAAAA063166400002FF)
```

In [ ]:
```
print(crc1, crc2, sep = '\n')
```

```
000000000000000
000000000000000
```

In [ ]:
```
print(data1, data2, sep = '\n')
```

```
000011000101100110010000
000110001011001100100000
```

We have gotten all the data from the frames, now let us see what happens when we check if our data is correct using the crc. The data is obviously different, so we should expect an error to occur.

In [ ]:
```
generating_poly_binary = '1100010110011001' # this is the binary represen
```

From the underlying principal of CRC, we expect to get a reminder of 0 when we divide the data with crc bits added by the generating polynomial

In [ ]:
```
print(int(data1 + crc1, 2) % int(generating_poly_binary, 2))
```

Out[ ]:
```
0
```

The original data gives zero as expected,

In [ ]:
```
print(int(data2 + crc2, 2) % int(generating_poly_binary, 2))
```

```
0
```

We expect to get a non zero answer here, because the data has been changed and errors have occured, but in this case our CRC fails to detect this and gives a reminder of 0. We want to find the likelihood of this happening.

In [ ]:
```
num_bits_changed = 0
a = data1 + crc1
b = data2 + crc1
for i in range(len(a)):
    if a[i] != b[1]:
        num_bits_changed +=1

print(num_bits_changed)
```

8

This case happens only when the data has been modified in a special way that it happens to fool the CRC. In this case it requires 8 bit swaps. So the chance that this happens is $10^{-2^8} = 10^{-16}$. That is there is a one in 10^{16} chance that this will happen.