

BLG433E

Computer Communications

2013 Fall



Report of Project 1

Date of Submission : 09.11 2013

Student Name: Volkan İlbeyli

Student Number : 040100118

Instructor : Sema Oktuğ

Program Related Explanations

The variables and data structures used are given below with the necessary explanations.

```
65  ///
66  /// INITIALIZE PROGRAM DATA
67  ///
68  vector<string> sender;           // 8 bits, read from file
69  vector<string> codedFrames;     // 12 bits, coded frames/codeword
70  vector<string> receiver;       // 12 bits, received erroneous frames
71  vector<string> correctedFrames; // 12 bits, corrected from the received ones
72  unsigned detectedErrors;
73  unsigned correctedErrors;
74  unsigned allErrors;
75  unsigned miscorrectedErrors = 0;
```

The main() has 3 parameters (except the function call itself, i.e. argv[0]): file name(string), error rate(double) and burst size(unsigned).

Compilation: g++ Main.cpp

Example Run: ./a.out input.txt 0.05 1

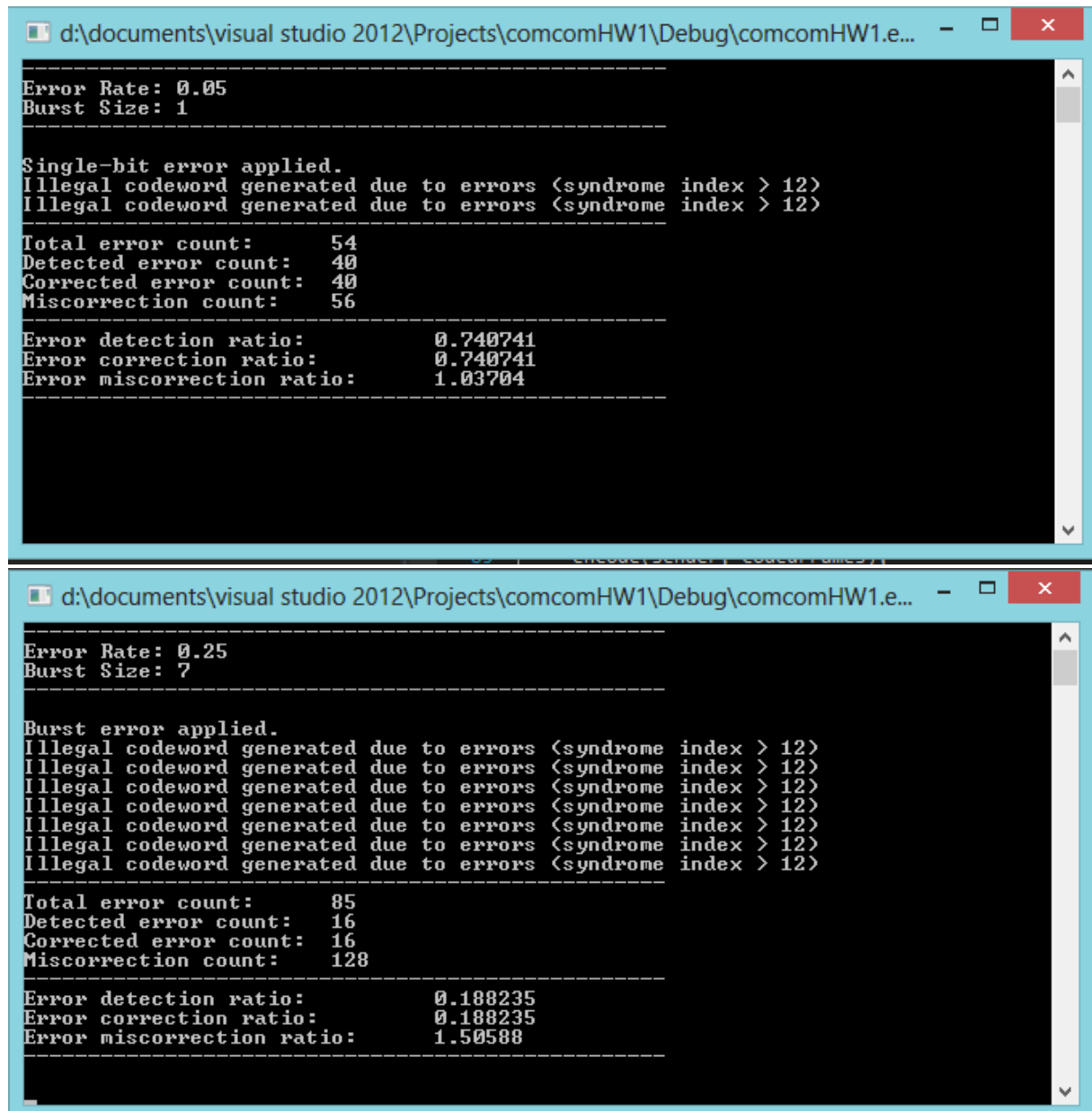
The sender is read from the file via the **readFile()** function and is 8-bits long. Then the **encode()** function is called and the parity bits are inserted and the codeword is stored in the codedFrames vector. Next, the **send()** function is called, where the environment conditions are applied to the transmission. Some bits are flipped according to the main() function parameters. If burst size is 1, for each bit that is transmitted, rand() is called and if the generated random number is smaller than errorRate parameter, the bit is flipped. If burst size is larger than 1 (it's also assumed its smaller than 12, no exception mechanism is implemented for burst size being larger than 12 situation), burst errors are implemented. For each frame, rand() function is called thrice. First call determines whether the burst error is going to happen in that frame or not according to the errorRate parameter. If the burst error occurs, second rand() call determines the index of the burst. The last (sequence of) rand() calls determines whether the bits in between the burst will be flipped or not, individually. Thus the receiver vector is populated.

After this point, the program runs the SEC algorithm on receiver via the **decode()** function. receiver's contents are copied into the correctedFrames vector and then the algorithm checks each frame's parity bits. By the Hamming Code conventions, for each parity bit (1,2,4,8), the corresponding bits are examined and the '1's are counted. If the '1' count is even, then the algorithm decides there are no errors. If the one count is not even, then parityCheck counter is incremented by the index of the parity bit. Therefore a syndrome index is generated via the parityCheck counter. Note that if there are multiple errors in such positions that could generate an illegal codeword, the algorithm will produce a segmentation fault. However in this case, "Transmission error occurred" message is outputted and no action is taken (no detection, no correction). If the parityCheck points to a legal index value (<12) then that bit is flipped and detectedError counter is incremented. Immediately after this, the sent codeword (error-free codeword that is stored in codedFrame vector) is compared with the corresponding corrected codeword that is stored in correctedFrame vector. If two vectors are different, than miscorrected bit counter is incremented by 8. No 8-bit data is decoded since the frames are compared.

Finally the **countErrors()** function is called which simply compares each codeword generated by sender with the received, erroneous codewords and counts the different bits. After these function calls, the program has the following calculated:

- Error detection count
- Error correction count (which is equal to detection since we use SEC algorithm)
- Error miscorrection count

Finally the program outputs the ratios.



The image shows two screenshots of a Visual Studio 2012 debug console window. The window title is "d:\documents\visual studio 2012\Projects\comcomHW1\Debug\comcomHW1.e...". The output is as follows:

```
-----
Error Rate: 0.05
Burst Size: 1
-----

Single-bit error applied.
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
-----
Total error count:      54
Detected error count:   40
Corrected error count:  40
Miscorrection count:    56
-----
Error detection ratio:      0.740741
Error correction ratio:    0.740741
Error miscorrection ratio: 1.03704
-----
```

The second screenshot shows the output for a burst error:

```
-----
Error Rate: 0.25
Burst Size: 7
-----

Burst error applied.
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
Illegal codeword generated due to errors <syndrome index > 12>
-----
Total error count:      85
Detected error count:   16
Corrected error count:  16
Miscorrection count:   128
-----
Error detection ratio:      0.188235
Error correction ratio:    0.188235
Error miscorrection ratio: 1.50588
-----
```

Discussion Section

- Discuss what can be an advantage and a drawback for having burst errors

Since burst errors contain multiple bits of errors, it becomes harder to correct those errors. Considering simple algorithms like SEC, it is impossible to correct this kind of error since a bit sequence is erroneous. The advantage is, on the other side, since the error rate is considered for the frames that are sent rather than the individual bits, fewer burst errors will occur than that of the single-bit errors (not counting the total flipped bit count, just the occurrences are considered).

- Discuss how we can use this code wisely for the case of burst errors

If the error rate is large for SEC to do better job in burst error mode than the single bit error mode, we may use this code. Charts below explain this better.

- What is the code rate for described hamming error detection mechanism

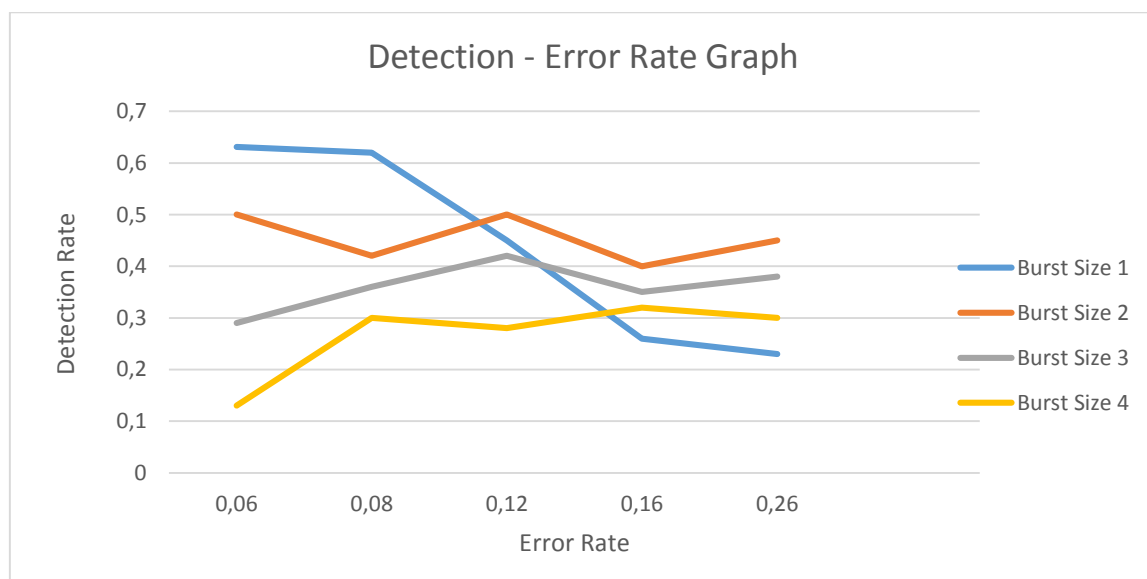
8 bits of data, 12 bit codeword → code rate is: $8/12 = 0.67$

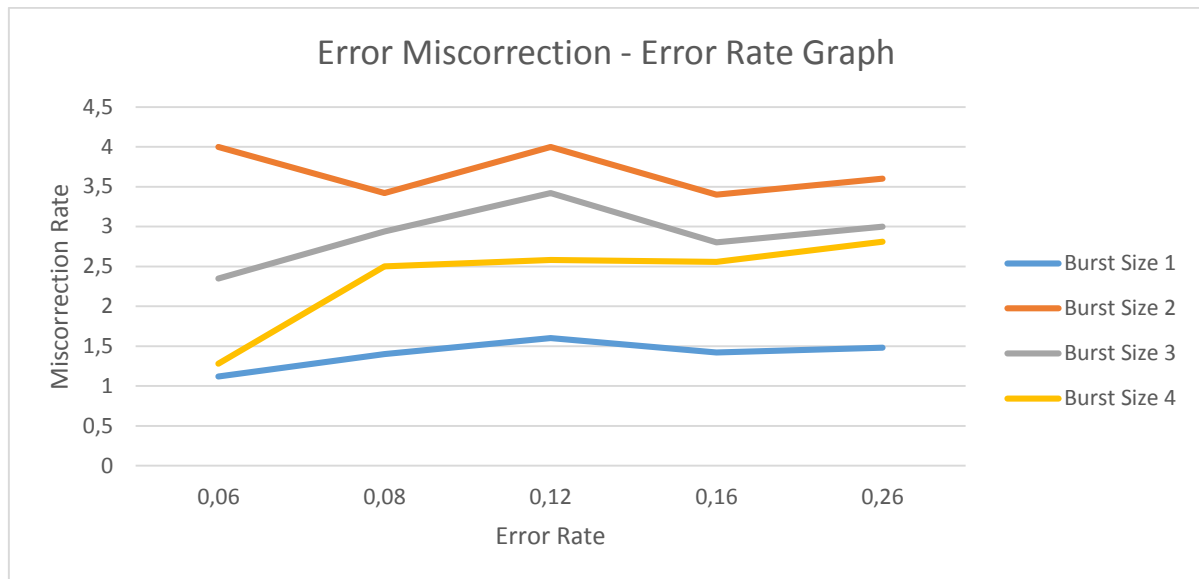
- What is hamming distance of the complete code (legal codeword set) you will use in this project?

Since we are using SEC algorithm, the hamming distance will be ($d=1$, $2d+1 = 3$) 3 in this project.

- In general, why you need distance $2d+1$ code to correct d errors?

Since $2d+1$ distance makes the legal words far from each other, when d errors occur, it is possible to correct the error “correctly” because the algorithm assumes the legal codeword is d distance away while the possible illegal one is $d+1$, thus chooses the right codeword for correction.





The first graph is the error detection vs error rate graph. When burst size is 1, i.e. single bit errors, we see the SEC algorithm does better job than burst errors on detection because burst error occurrence means multiple errors (at least 2, at most burst size) and thus lower error detection. However as the error rate increases, we see the burst errors advantage over single bit errors as the error detection rate of the SEC algorithm drops below the burst error detection rates. Error correction is the same graph with the error detection due to SEC only being able to correct the detected errors. Therefore no extra chart is graphed.

The second graph shows the error mis-correction rate vs error rate. The lower the mis-correction rate the better it is. Yet again, we see SEC does better job in single errors than burst errors due to it being able to detect and correct single errors. As the error rate increases, the mis-correction rate tends to increase in both error modes. Concluding from both graphs, SEC is a better choice for little erroneous environments.