TITLE

Name: Sayan Acharya

Class: BCSE-2024

Group: A1

Roll number: 002010501009

Assignment number: 1

Problem Statement: Error Detection using CRC, VRC, LRC, Checksum

Date: 12/08/2022

Design

While sending data from a sender to a receiver, due to various noise induced, the databits can get randomly flipped. For the detection of this error, 4 different schemes are introduced. They are:

1. LRC (Longitudinal Redundancy Check)

2. VRC (Vertical Redundancy Check)
3. Checksum
4. CRC (Cyclic Redundancy Check)

For the designing of the module, I have built a python module consisting of different files where each detection method is kept ina different file and consists of various helper and main methods for the detection of the error.

The files considered for this assignment are:

1. Misc.py
2. CheckSum.py
3. CyclicRedCheck.py
4. LongRedCheck.py
5. VertRedCheck.py
6. main.py
7. sender_receiver.py

- Misc.py contains helper functions and exceptions that are used in the other files
- Checksum.py contains methods for checksum validation, checksum redundancy building and detection
- CyclidRedCheck.py contains methods that keeps the polynomial, division function and validation function
- LongRedCheck.py contains methods for calculation of packet lengthwise numbers and then xoring them
- VertRedCheck.py contains methods for the calculation of the parity of the string and validation of the redundancy.
- main.py contains a driver code to run all the methods and test them with various files. It also has the answer to the 3 questions mentioned in the assignment
- sender_receiver.py contains sender, receiver classes that just send, receive, encode data

Method Description

Misc.py: This file contains 2 exceptions namely:

- LengthMismatchException: this exception is thrown when length of the bitstring is not a multiple of 4/8/packet_length
- InvalidBinaryString: this exception is thrown when the string contains characters other than 0 or 1.

and 1 string validation function:

- validate_binary_string that validates whether the string is binary or not and if not validated

All the other error detection classes have same named functions but the implementations vary. I will describe them in general here and for the special changes classwise, i will describe those in points

- validate_string: This function checks whether the given string is a valid binary string wih the right length or not. For example, for LRC the string has to be a multiple of packet_length. etc
- get_redundancy: This function gives the redundanyt bits to e added at the back of the original data.
- create_redundant_frame: This function concatenates the original data and redundant frame and returns the output string
- is_valid_frame: This checks whether the given redundancy-added data is a valid data or not.

Mainly these 4 methods are there and apart from that a __str__ method is described for each class to help with the debugging. Also an __init__ function is there.

LongRedCheck.py:

- validate_string function of this class checks if the string is binary and then checks if the dara is a multiple of packet_length
- create_redundant_frame function of this class xors the longitudinal arts of the data and returns that as the redundant frame

VertRedCheck.py:

- validate_string function of this class just checks if the string is binary or not
- create_redundant_frame of this class just checks the parity of the given string. For that purpose, the binary values of the string are just xored together. This is returned as the redundant bit

CheckSum.py:

- create_redundant_frame function of this class takes all the input data, divides them into packets of packet_length and then adds them up. Any extra carry after the packet_length bits are wrapped back. then this summatio value is 1's complemented and that is returned as the redundant frame of the method.
- is_valid_frame function of this class applies the smae operation described above and checks whether the output is a string of 0s or not

CyclicRedCheck.py:

- create_redundant_frame of this class divides the given string with a pre fixed polynomial and then the remainder of the operation is taken as the redundant frame.
- is_valid_frame functions takes the whole data(input+redundancy) and applies the same division on this. If the result is a string of 0s then the frame is valid

main.py:

This file contains the driver code to test the given classes and also the answers to the 3 questions of the assignment

sender_receiver.py:

This file contains the sender receiver classes and those classes contain methods like encode,decode, error_check etc.
The code for these files are provided separately.

Testcases

1st Question:

The input data was corrupted using an injection function that randomly injects errors in the given string.

```
input_data: 110101010
error_data: 010101010
Error detected by LongRedCheck: True
Error detected by VertRedCheck: True
Error detected by CheckSum with packet_len=4: True
Error detected by CyclicRedCheck with poly 11001: True
```

2nd Question:

The crc failed here cause the corrupted bits were at least 7 bits apart and the input polynomial is a multiple of the pre fixed divider polynomial

```
input_data: 10000001
error_data: 00000000
Error detected by CheckSum with packet_len=4: True
Error detected by CyclicRedCheck with poly 1101: False
```

3rd Question:

This was a tricky one. The vrc can only detect the error if the number of wrong bits is odd and crc can detect any error with only 1 bit changed. So for this, the input data was corrupted in a way so that the difference between the changed bits was more than 7 and less than 16 and also the given polynomial is a multiple of the pre fixed polynomial.

```
D:\NetworkLab\Assignment1>python -u "d:\NetworkLab\Assignment1\main.py"
input_data: 10101101011110001110101111000101101011000110001
error_data: 10101101011010001010101101000101101011000110001
Error detected by VertRedCheck: True
Error detected by CyclicRedCheck with poly 11001: False
```

Results

LRC.py:

Suppose the given data is divided is like this: 1000,0100,1011.
Then the LRC redundancy of these frames are:

1000
0100
1011
—————————
0111

The 3 frames are simply xored and the output is strapped together with the data

VRC.py:

Suppose the input data is 1011 then the redundancy will be 1^0^1^1=1. This extra bit will be strapped at the back of the data and that will be the whole string

CheckSum.py:

Suppose the input data is like this: 1000, 0110, 1001.
Then they will be added togther:

1000
0110
1001
————————————
1 0111
————————————
0111
0001
————————————
1000

1's complement

—----------

0111

This extra 0111 will be concatenated with the given string and when validating the same process will be followed for the whole data. If the result comes out as 0000 then the string is validated.

CRC:

For this method a suitable polynomial is chosen and the given data is divided with that polynomial. Suppose the chosen polynomial is x^4+x^3+1. then the fixed divisor polynomial is 11001.

Now suppose the given data is: 1100110000

1100110000

11001

—----------

10000

11001

—----------

01001

So the redundant polynomial is 1001. These extra bits will be added to the given data and during validation the same process will be run and if the resultant remainder is 0000 then the string is validated

Analysis

LRC:

The LRC works well when there are errors in the different columns of the given data. It will be able to detect all such errors. If the errors are in the same coumn then only odd number of errors can be detected.

VRC:

The VRC only keeps the parity of the data and therefore will only be able to detect odd number of errors.

CheckSum:

This can detect errors when 2 bits of the same column do not get flipped. This what makes it a very robust error detection algorithm. Also it is very easy to implement and apply.

CRC:

This is the one of the most powerful error detection algorithms that can detect almost all errors. All 1 bit errors are detected and multibits errors are detected the dividend polynomial is not a multiple of the divider polynomial. And in the case of the multiple, errors spanning less than 7 bits are easily detected. Keeping all of this in mind, this method has the greatest chance to detect errors.

Comments

This assignment taught me how to implement and apply various error detection algorithms and what are the strengths and weaknesses of various algorithms. It was an enjoyable experience overall.