ERROR DETECTION METHODS (CRC, VRC, LRC, CHECKSUM) USING SOCKET PROGRAMMING

Nikhil Mahajan

Department Of Information Technology

Vishwakarma Institute Of Technology, Pune,411037, Maharashtra, India

ABSTRACT

The analogue signals in digital systems transform into a digital sequence. Data stream refers to this collection of bits. The thesis of this study asserts that a single bit's location can shift and still cause a significant inaccuracy in the data output. We provide an overview of error control in this work, including error detection and error repair. It indicates that in order to provide a precise or approximative result, error detection and correction techniques are used to identify errors. The network's handling and detection of mistakes, particularly at the data connection layer, is referred to as error control. This article focuses on the various error detection algorithms used to identify mistakes and how to fix them so the receiver can extract the real data.

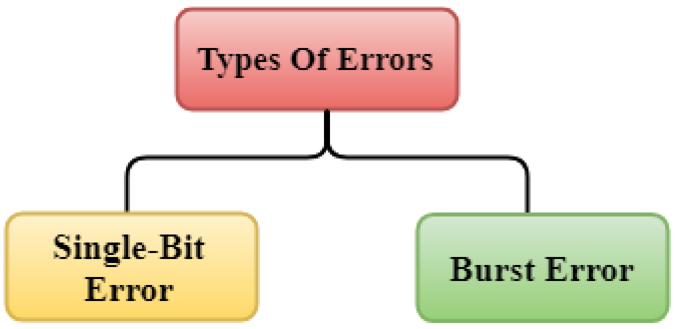
KEYWORDS

Types of Error, Error Detection, Error Correction Techniques

INTRODUCTION

A bit, on travel, is subjected to electromagnetic interference due to noises. Thus, the data transmitted may be prone to errors. In easy words, any bit can be flipped. This happens because the voltage present in noise signals either has direct impact on the voltage present in the data signal or creates a distortion leading to mis-interpretation of bits while decoding the signals at the receiver. This calls for a study on error detection and error correction. The receiver must be capable enough to detect the error and ask the sender to retransmit the data packet.

TYPES OF ERRORS



[1] [1, 2, 3, 4]

Errors can be classified into two categories:

Single-Bit Error

Burst Error

Single-Bit Error:

The only one bit of a given data unit is changed from 1 to 0 or from 0 to 1.

Single-Bit Error does not appear more likely in Serial Data Transmission. For example, Sender sends the data at 10 Mbps, this means that the bit lasts only for 1?s and for a single-bit error to occurred, a noise must be more than 1 ?s.

Single-Bit Error mainly occurs in Parallel Data Transmission. For example, if eight wires are used to send the eight bits of a byte, if one of the wire is noisy, then single-bit is corrupted per byte.

Burst Error:

The two or more bits are changed from 0 to 1 or from 1 to 0 is known as Burst Error.

The Burst Error is determined from the first corrupted bit to the last corrupted bit.

The duration of noise in Burst Error is more than the duration of noise in Single-Bit.

Burst Errors are most likely to occur in Serial Data Transmission.

The number of affected bits depends on the duration of the noise and data rate.

ERROR DETECTION METHODS

Following are the error detection methods or techniques of error detection in networking.

1. VRC Method

Every data unit in this method of error detection has a redundant bit called a parity bit added to it so that the total number of 1s in the unit (including the parity bit) is even. The system is now using the network link to transmit the complete extended unit. All eight received bits are examined at the receiver using an even parity checking procedure. Even the data unit of 1 is valid if it counts. If it counts an odd number of ones, there has been some sort of data error. Receiver rejects the entire data unit as a result. The implementation of odd parity VRC is also possible. The total number of 1s used in this manner should be odd before transmission.

2. LRC METHOD

A block of bits are arranged in a table as part of this error detection technique (of rows and columns). For instance, before delivering a 32-bit block, it is first divided into four rows and eight columns. Then a fresh row of eight parity bits is produced after calculating the parity bits for each column. The original data is prefixed with these eight parity bits before transmitting.

3. CRC METHOD

An error-detecting code called a cyclic redundancy check (CRC) is frequently used in digital networks and storage devices to find unintentional modifications to digital data. In these systems, each block of data that enters receives a short check value depending on the remaining polynomial division of its contents. The calculation is performed upon retrieval so that corrective action against data corruption can be taken if the check values do not coincide. CRCs can be applied to repair errors. CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyse mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

4. CHECKSUM METHOD

There are two modules in this error detection method viz. checksum generator and checksum checker. In the transmitter, checksum generator subdivides data unit into equal segments of n bits (usually 16). These segments are added together using one's complement arithmetic in such a way that total is also n bits long. The total (i.e., sum) is then complemented and appended to the end of the original data unit as redundancy bits, called checksum field.
The receiver subdivides the data unit as above and adds all segments together and complements the result. If the extended data unit is intact, the total value found by adding all data segments and checksum field should be zero. If the result is not zero, the packet contains error and receiver rejects the packet.

PROGRAM FILES

* server.py : data receiver * client.py : data sender * lrc.py, vrc.py, checksum.py, crc.py : contains the classes of respective error detection methods. * helper.py : definitions of basic CRC methods The individual files fulfils different assignment purposes, following which have been explained in details : 1. Client.py: The following are the tasks performed in this Client program: a. Takes a string from stdin from user. b. Converts the string into binary data stream; and then breaks the stream into datawords of length 8. c. In case of LRC, VRC, the calculated redundant bits forms a packet and sent separately, for checksum, the calculated 1s complement sum is sent separately. d. In case of CRC, the redundant bits of remainder are added to each dataword and the respective codewords are sent through sockets. e. Before sending data, some errors are injected at random positions. e. Lastly, it

receives a message from server if the data received was erroneous or not.

2. Server.py: The following are the tasks performed in this Server program: a. After connecting to the client, server receives the total data size, method of error detection, and all the codewords/datawords until EOF is received through sockets. b. Checks for errors based on the method specified. c. Sends the client a message indicating the status of received data.

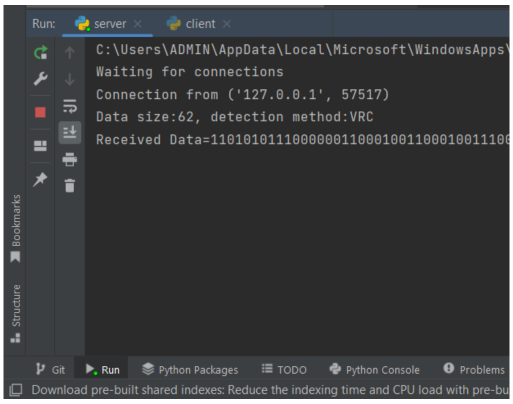
3. lrc.py, vrc.py, checksum.py, crc.py: Contains classes for respective error detection methods which contains both encoding and decoding methods for a list of datawords.

4. Helper.py: Contains the definitions of types of CRC methods mapped to respective polynomials and a function to convert those polynomials into binary divisors.

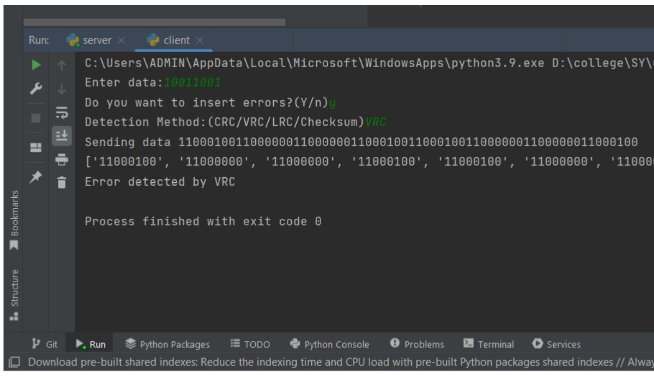
RESULTS

For each type of error detection methods there are certain advantages as well as disadvantages discussed below:

a. Vertical Redundancy Check (VRC): Since parity for each data words are calculated separately, it can detect any single bit errors as well as odd number of burst errors, but can't detect any even number of burst errors, for example: The dataword:11101001 has 5 1's hence VRC is 1, if erroneous data word be like: 11111111, then it can easily detect the error as it has 8 1's. But if the errorneous dataword be like: 11101111, it will not detect errors as it has 7 1's.

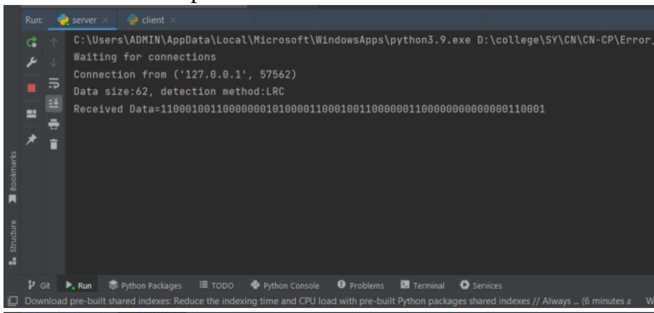


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\
Waiting for connections
Connection from ('127.0.0.1', 57517)
Data size:62, detection method:VRC
Received Data=1101010111000000110001001100010011100
```

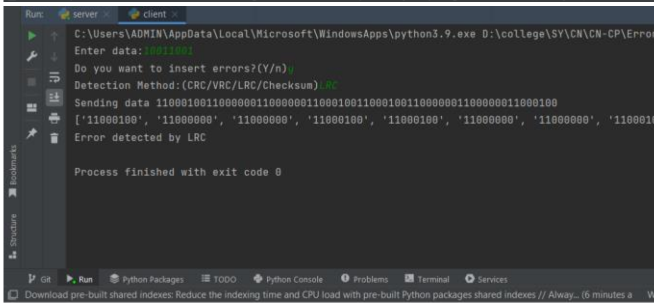


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\college\SY\
Enter data:10011001
Do you want to insert errors?(Y/n)y
Detection Method:(CRC/VRC/LRC/Checksum)VRC
Sending data 11000100110000001100000011000100110001001100000011000000110001OO
['11000100', '11000000', '11000000', '11000100', '11000100', '11000000', '11000

Error detected by VRC

Process finished with exit code 0
```

b. Longitudinal Redundancy Check (LRC): There may be 2 cases where LRC cannot detect errors, if 2 datawords have flipped bits in the same position like 1110 and 1001 becomes 1111 and 1000, for the last bit LRC remains same; and another case would be if even number of datawords have error in the same position.

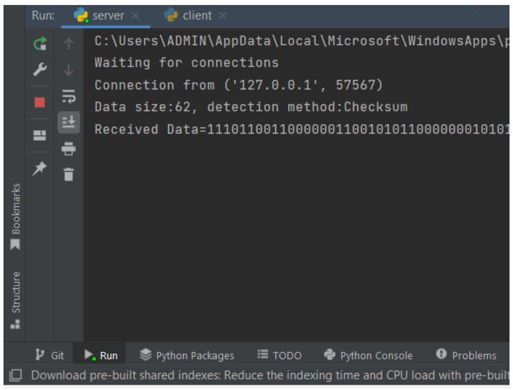


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\college\SY\CN\CN-CP\Error_
Waiting for connections
Connection from ('127.0.0.1', 57562)
Data size:62, detection method:LRC
Received Data=11000100110000000101000011000100110000001100000000000000110001
```

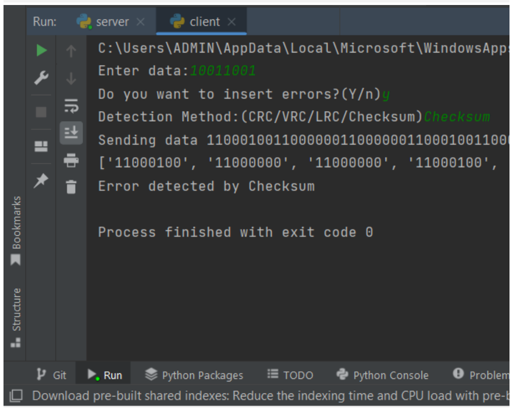


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\college\SY\CN\CN-CP\Erro
Enter data:10011001
Do you want to insert errors?(Y/n)y
Detection Method:(CRC/VRC/LRC/Checksum)LRC
Sending data 11000100110000001100000011000100110001001100000011000000110001OO
['11000100', '11000000', '11000000', '11000100', '11000100', '11000000', '11000000', '110001
Error detected by LRC

Process finished with exit code 0
```

c. Checksum: Checksum may also have 2 types of cases where the 1's complement sum remains the same but there are errors in the sent data: i) If bits in the same position are flipped, for example: 1110 and 1001 has checksum 0111 but 1111 and 1000 also has the same checksum. ii) If two bits having 0 are flipped to 1 and in the more significant bit a 1 is turned to 0, the sum will still remain the same.

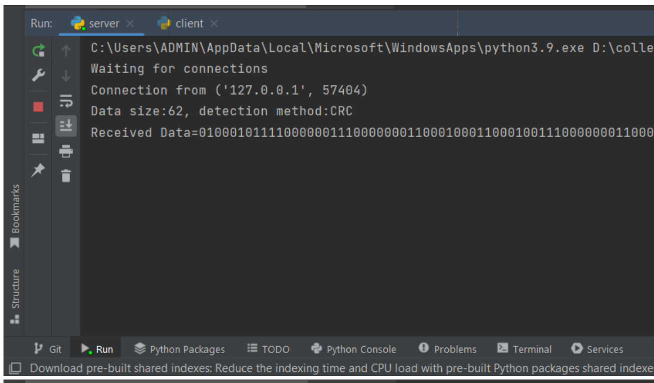


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\p
Waiting for connections
Connection from ('127.0.0.1', 57567)
Data size:62, detection method:Checksum
Received Data=11101100110000001100101011000000010101
```

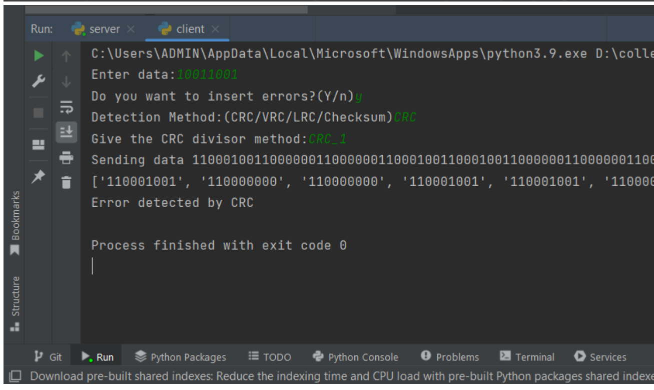


```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps
Enter data:10011001
Do you want to insert errors?(Y/n)y
Detection Method:(CRC/VRC/LRC/Checksum)Checksum
Sending data 11000100110000000110000000110001001100
['11000100', '11000000', '11000000', '11000100',
Error detected by Checksum

Process finished with exit code 0
```

d. Cyclic Redundancy Check (CRC): The answer to whether CRC detects all errors depends on the divisor polynomial used as a part of CRC computation. Consider the divisor polynomial x^2 which corresponds to 1 0 0 and the message to be transmitted is 101010. After appending CRC bits (in this case 0 0) get 10101000. Assuming during the data transmission, the 3 bit is in error and the message received at the receiver is 10101100. On performing CRC check on 10101100, we see that the remainder is zero and the receiver wrongly concludes that there is no error in transmission. The reason this error is undetected by the receiver is that the message received is perfectly divisible by the divisor 1 0 0. More appropriately, if we visualize the received message as the xor of the original message and the error polynomial, then we see that the error polynomial is divisible by 1 0 0. However, if the divisor is 1 0 1, then both errors are detected by the receiver. Thus, choosing an appropriate divisor is crucial in detecting errors at the receiver if any during the transmission.



```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\colle
Waiting for connections
Connection from ('127.0.0.1', 57404)
Data size:62, detection method:CRC
Received Data=0100010111100000011100000001100010001100010011100000000110000
```



```
C:\Users\ADMIN\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\colle
Enter data:10011001
Do you want to insert errors?(Y/n)y
Detection Method:(CRC/VRC/LRC/Checksum)CRC
Give the CRC divisor method:CRC_1
Sending data 1100010011000000011000000011000100110001001100000001100000001100
['110001001', '110000000', '110000000', '110001001', '110001001', '11000
Error detected by CRC

Process finished with exit code 0
```

CONCLUSSION

a. Error detection capabilities of the code is increased significantly when all 4 schemes are used. Approximately 3.2% of the time, error is not detected by any of the schemes.

b. The errors introduced are random and erratic, so the results obtained are subject to experimental errors. There can be cases where a same bit is flipped an even number of times and hence no error is introduced by the inject error () function. Such cases have been ignored and the program assumes that the function is indeed able to introduce errors in all codewords.

c. Since sockets are used for data transfer, the incomplete transfer of socket buffers can cause some undesired results, hence before sending each data element, the process is suspended for 1second, which again slows down the whole process.

References

[1] A. Azahar, R. Alsaqour, M. Al-Hubaishi and M. Uddin, "Review of Error Detection of Data Link layer in Computer Network," *Middle-East Journal of Scientific Research*, vol. 18, no. 7, 2013.

[2] M. Roshanzadeh and S. Saqaeeyan, "Error Detection & Correction in Wireless Sensor Networks By using Residue Number Systems," *I. J. Computer Network and Information Securit*, no. 2, 2012.

[3] J. S. Sobolewski, Cyclic redundancy check, 2003.

[4] Y. Zhang and Q. Yuan, "A multiple bits error correction method based on cyclic redundancy check codes," *9th International Conference on Signal Processing*, 2008.