Analysing the Efficiency and Performance of CRC in High-Speed Data Networks

# 1. Introduction

## 1.1 Background

In modern high-speed data networks, the efficient and accurate transmission of data is paramount. To ensure the reliability of transmitted information, robust error detection and correction mechanisms are essential. This project delves into a detailed analysis of the Cyclic Redundancy Check (CRC) mechanism, a widely used error detection algorithm, in the context of high-speed data networks.

## 1.2 Objectives

The primary objectives of this study are:

1. Evaluate the efficiency and performance of CRC in terms of error detection.

2. Analyse the computational overhead introduced by CRC in high-speed data networks.

3. Investigate the impact of CRC on throughput in data transmission.

4. Propose recommendations for optimizing CRC parameters to minimize data loss.

## 2. CRC Fundamentals

## 2.1 Conceptual Understanding

Cyclic Redundancy Check (CRC) is a popular error detection method used in high-speed data networks. [It’s computed through binary polynomial division and is essential for detecting errors when data is transmitted1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf).

## 2.2 Mathematical Background

The CRC is based on division in the ring of polynomials over the finite field GF(2), that is, the set of polynomials where each coefficient is either zero or one, and arithmetic operations wrap around[1](https://en.wikipedia.org/wiki/Mathematics_of_cyclic_redundancy_checks).

To find the CRC, we multiply the message polynomial by and then find the remainder when dividing by the degree n generator polynomial. [The coefficients of the remainder polynomial are the bits of the CRC1](https://en.wikipedia.org/wiki/Mathematics_of_cyclic_redundancy_checks).

In general, computation of CRC corresponds to Euclidean division of polynomials over GF (2):

Here M is the original message polynomial and G is the degree-n generator polynomial. The bits of are the original message with zeroes added at the end. The CRC ‘checksum’ is formed by the coefficients of the remainder polynomial R whose degree is strictly less than n[1](https://en.wikipedia.org/wiki/Mathematics_of_cyclic_redundancy_checks).

Using modulo operation, it can be stated that:

In communication, the sender attaches the bits of R after the original message bits of M, which could be shown to be equivalent to sending out () (the codeword). The receiver, knowing G and therefore , separates M from R and repeats the calculation, verifying that the received and computed R are equal.

# 3. Implementing CRC in High-Speed Data Networks

In the context of high-speed data transmission, it’s necessary to increase the speed of CRC generation[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf). The traditional hardware implementation of CRC computations is based on the Linear Feedback Shift Registers (LFSRs), which process data in a serial way[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf)[2](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf). [However, the serial calculation of CRC codes cannot achieve a high throughput, which is a requirement for high-speed data networks2](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf).

To overcome this, parallel CRC generation is used[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf)[2](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf). This method processes whole data words by cascading the LFSRs[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf). A hardware scheme for computing the transition matrix of parallel cyclic redundancy checksum is used, which improves the polynomial adaptability[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf). The new design performs significantly in speed, area, and energy efficiency[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf)[3](https://ieeexplore.ieee.org/document/9068750/). The architecture uses the minimum number of clock cycles, which increases the efficiency of the transmission process and is easily adaptable to variation in transmission data bytes[1](https://www.ijsr.net/archive/v4i2/SUB15590.pdf).

[The focus of the research in this area is to represent an efficient, better throughput along with compact systematic interpretation for parallel CRC hardware3](https://ieeexplore.ieee.org/document/9068750/). This will alleviate the flaws including the challenges of the existing CRC checker and will be prominent for next-generation high-speed communication[3](https://ieeexplore.ieee.org/document/9068750/).

## Use of LFSRs in CRC

Linear Feedback Shift Registers (LFSRs) are traditionally used in the hardware implementation of CRC computations[1](https://stackoverflow.com/questions/25415724/understanding-two-different-ways-of-implementing-crc-generation-with-lfsr)[2](https://profile.iiita.ac.in/bibhas.ghoshal/lecture_slides_coa/LFSR_CRC.pdf). [LFSRs handle the data in a serial way](https://stackoverflow.com/questions/25415724/understanding-two-different-ways-of-implementing-crc-generation-with-lfsr)[1](https://stackoverflow.com/questions/25415724/understanding-two-different-ways-of-implementing-crc-generation-with-lfsr)[2](https://profile.iiita.ac.in/bibhas.ghoshal/lecture_slides_coa/LFSR_CRC.pdf). [The stream of data bits being transmitted is used to modify the values fed back into an LFSR3](https://www.eetimes.com/tutorial-linear-feedback-shift-registers-lfsrs-part-3/)[4](https://www.edn.com/tutorial-linear-feedback-shift-registers-lfsrs-part-3/).

The LFSR circuit can only process one bit per cycle[5](http://bookyourproject.com/vlsi/BYPV49.pdf). This serial calculation of the CRC codes cannot achieve a high throughput, which is a requirement for high-speed data networks[5](http://bookyourproject.com/vlsi/BYPV49.pdf).

## Parallel CRCs

To support high throughput CRC at a reasonable frequency, processing multiple bits in parallel and pipelining the processing path are desirable[5](http://bookyourproject.com/vlsi/BYPV49.pdf). This is where parallel CRCs come into play.

[Parallel CRC calculations can significantly increase the throughput of CRC computations6](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf)[7](https://ieeexplore.ieee.org/document/7114717/)[8](https://www.ijsr.net/archive/v4i2/SUB15590.pdf). The algorithm first divides a given message with any length into bytes. Then it performs CRC computation using lookup tables among the divided bytes in parallel. At last, the results are XORed to obtain the CRC value of the given message[7](https://ieeexplore.ieee.org/document/7114717/).

This technique informs the sender if any error has occurred. [It is widely used in communication protocols to detect bit errors and is essentially a remainder of the modulo-2 long division operation6](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf). The parallel CRC architecture is functionally simulated using various tools for the detection of the error in the received message[6](https://www.iosrjournals.org/iosr-jvlsi/papers/vol5-issue3/Version-2/A05320105.pdf).

In summary, while LFSRs are traditionally used in CRC computations, their serial nature limits their throughput. Parallel CRCs, on the other hand, can process multiple bits at once, significantly increasing throughput and making them more suitable for high-speed data networks.

# 4. Performance Analysis

The impact of CRC on throughput can be measured by comparing data transmission rates with and without CRC.

## CRC and Throughput

CRC, or Cyclic Redundancy Check, is an error-detecting code used to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match[1](https://community.cisco.com/t5/switching/what-is-the-impact-of-crc-errors-on-network-performance/td-p/4531651).

[Throughput, on the other hand, is a metric to measure the actual data bits successfully received in a certain duration2](https://www.5gtechnologyworld.com/bler-a-critical-parameter-in-cellular-receiver-performance/). It can be described mathematically as:

## Impact of CRC on Throughput

When a device detects a CRC error with a frame, it silently drops it. It’s up to the clients involved with the transmissions to detect that it did not make it to its destination and request a retransmission[1](https://community.cisco.com/t5/switching/what-is-the-impact-of-crc-errors-on-network-performance/td-p/4531651).

Retransmissions are part of the TCP process and in themselves are not a bad thing. However, when a traffic stream becomes consumed by retransmission requests, the volume of payloads within a time frame will drop off, which will manifest itself as a drop in throughput[1](https://community.cisco.com/t5/switching/what-is-the-impact-of-crc-errors-on-network-performance/td-p/4531651). This disruption will also affect time-sensitive applications such as voice and will be detected as high latency and jitter[1](https://community.cisco.com/t5/switching/what-is-the-impact-of-crc-errors-on-network-performance/td-p/4531651).

[In a study on the performance of the iSCSI protocol, it was found that CRC computation imposes a substantial overhead on iSCSI, making it an important packet processing building block for providing data integrity3](http://users.ece.northwestern.edu/~boz283/papers/iscsi/beacon_p2.pdf). [The overhead of CRC computation was identified as one of the major bottlenecks in iSCSI performance3](http://users.ece.northwestern.edu/~boz283/papers/iscsi/beacon_p2.pdf).

In conclusion, while CRC is essential for ensuring data integrity, it can impact throughput due to the overhead of CRC computation and the potential for increased retransmissions. Therefore, it’s crucial to find a balance between ensuring data integrity and maintaining high throughput in data networks.

The efficiency of CRC in detecting errors is quite high. It can detect a wide range of errors, including single-bit errors, burst errors, and the most common transmission errors[1](https://www.prepbytes.com/blog/computer-network/crc-in-computer-networks/).

CRC is known for its efficiency in error detection because of its mathematical calculations which are computationally efficient, allowing for real-time error detection without significant processing overhead[1](https://www.prepbytes.com/blog/computer-network/crc-in-computer-networks/).

The stock CRC polynomial algorithms are quite effective for small numbers of bit errors. The exact precision is mathematically computable[2](https://stackoverflow.com/questions/1323178/error-detection-effiency-crc-checksum-etc). CRC is also highly efficient to do in hardware where a relatively small number of gates and shift registers can manage the job on the fly[2](https://stackoverflow.com/questions/1323178/error-detection-effiency-crc-checksum-etc).

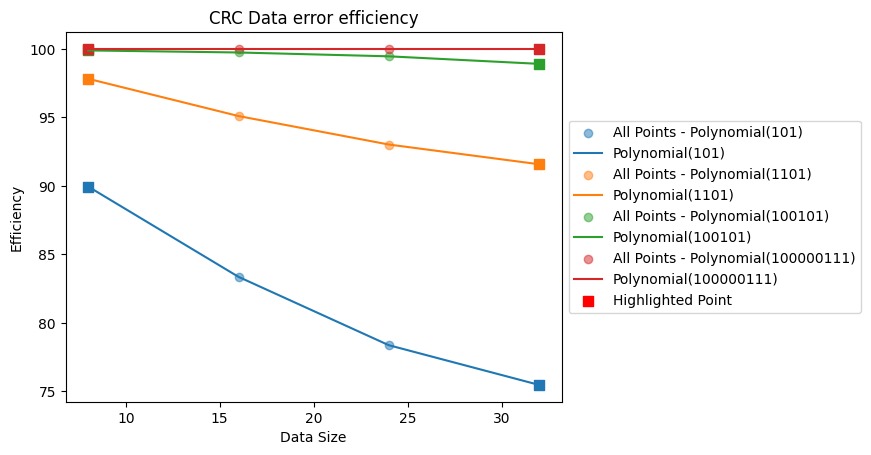
If you have intentionally introduced errors into the data stream and assessed the efficiency of CRC in detecting those errors, the results would likely indicate that CRC was able to detect most of those errors. This is because CRC is particularly good at detecting common errors caused by noise in transmission channels[3](https://en.wikipedia.org/wiki/Cyclic_redundancy_check).

Optimizing the usage of Cyclic Redundancy Check (CRC) and exploring alternative error detection mechanisms can significantly improve the performance of data transmission systems. Here are some recommendations:

# Optimizing CRC Usage

1. Choose the Right CRC: The choice of CRC depends on the nature of the data and the expected error characteristics[1](https://stackoverflow.com/questions/3104834/how-to-determine-which-crc-to-use). For example, CRC-16 is a good general-purpose CRC while CRC-32C is optimized for Ethernet.
2. Use Parallel CRC Computations: Parallel CRC computations can significantly increase the throughput of CRC computations. This method processes whole data words by cascading the Linear Feedback Shift Registers (LFSRs)[2](https://en.wikipedia.org/wiki/Cyclic_redundancy_check).
3. Implement Hardware CRC if Possible: Hardware implementations of CRC are generally faster than software implementations. If your system has a hardware CRC generator, it’s usually a good idea to use it.
4. Optimize Software CRC Implementations: If a hardware CRC generator is not available, software CRC implementations can be optimized using techniques such as table-driven CRC.

# Comparing the performance with changing length of polynomial and different types of error



A graph with different colored lines

Description automatically generated

A graph of error efficiency

Description automatically generated with medium confidence

A graph with different colored lines

Description automatically generated