Analysis and applications of various checksum based error detective algorithm

Member-1 Harsha V, Member-2 Satya sai prakash, Member-3 Tejus V S
Department of Computer Science and Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore, India
Mobile No.-1 8951053371, Mobile No.-2 7095870931, Mobile No.-3 9113062535
harshav.211cs129@nitk.edu.in, kodisatyasaiprakash.211cs133@nitk.edu.in, tejusvs.211cs159@nitk.edu.in

July 21, 2023

INTRODUCTION

0.1 Background of the problem statement

A checksum [1–3]is a technique of error detection that is created when all the bytes or words are summed up in a data word. The checksum is attached to and transmitted with it, making this a systematic code in which the data being sent is included in the code word unchanged.Network receivers again compute the checksum of the received data word and compare it to the received checksum value.If there's a match between computed and received codeword, then it is not likely that the message suffered a transmission error. The XOR , 1's complement,2's complement are the least effective checksums and take very less computational cost. The most expensive commonly used checksum is a CRC .we analyse various checksum methods and their positives and negatives and probability of undetected error and it's effects.

0.2 issues of problem statement

While checksums can be useful for detecting errors in digital data, they are not without their issues. Here are some common issues associated with checksums:

Limited error detection: Checksums can only detect certain types of errors, such as single-bit errors and some multiple-bit errors. However, they may not be able to detect more complex errors, such as those caused by multiple-bit errors that cancel each other out, or errors that occur in specific patterns.

False positives: Sometimes, a checksum may indicate that data is corrupted when it is actually fine. This can happen if the checksum is not calculated properly, or if the data itself has a pattern that matches the checksum. False positives can lead to unnecessary data retransmissions or even data loss.

Vulnerable to intentional tampering: Checksums are not secure against intentional tampering. Attackers can modify the data and then recalculate the checksum to match the new data, making it difficult to detect the tampering.

Overhead: Calculating and verifying checksums adds extra overhead to data transmission or storage, which can reduce overall performance.

Limited error correction: Checksums are designed to detect errors, but they cannot correct them. In situations where errors are common, more advanced error correction techniques, such as forward error correction (FEC), may be necessary.

0.3 exsisting approaches and their issues

1.XOR checksum issues: Limited error detection: Like other types of checksums, XOR checksums can only detect certain types of errors, such as single-bit errors and some multiple-bit errors. They may not be able to detect more complex errors, such as those caused by multiple-bit errors that cancel each other out, or errors that occur in specific patterns.

Limited security: XOR checksums are not secure against intentional tampering. Attackers can modify the data and then recalculate the checksum to match the new data, making it difficult to detect the tampering.

Overhead: Calculating and verifying XOR checksums adds extra overhead to data transmission or storage, which can reduce overall performance.

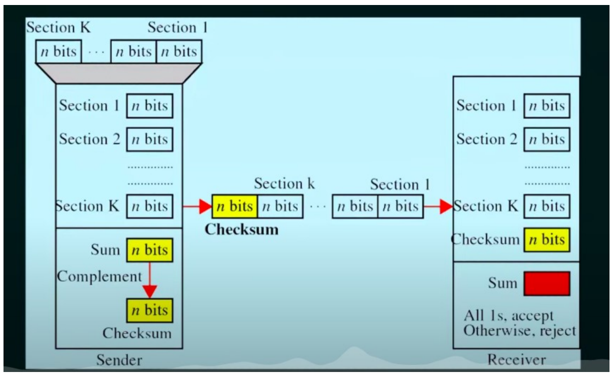


Figure 1: checksum

Limited error correction: Like other types of checksums, XOR checksums are designed to detect errors, but they cannot correct them. In situations where errors are common, more advanced error correction techniques, such as forward error correction (FEC), may be necessary.

Sensitivity to bit order: XOR checksums are sensitive to the order of the bits in the data being checked. This means that the same data in a different order may result in a different checksum value, even if the data is actually the same. This can lead to false positives and other errors. 2. 2's complement checksum issues: Limited error detection: While the 2's complement checksum can detect errors such as a flipped bit or a missing byte, it cannot detect errors such as the transposition of two adjacent bytes or the insertion of an extra byte.

False positives: It is possible for the 2's complement checksum to produce a checksum value of zero even if there are errors in the data. This can happen if the errors happen to cancel each other out during the checksum calculation.

Limited error correction: The 2's complement checksum method can only detect errors, not correct them. If errors are detected, the entire message must be retransmitted, which can be time-consuming and inefficient. 3. 1's complement checksum same issues as the 2's complement checksum Similarly we have 1's complement fletcher checksum,Adler checksum,CRC which have got some issues we can find

0.4 problem statement

dilemma about which checksum to use are difficult because of a less information about the relative effectiveness of available options. This problem mainly deals with which checksum method to use when we have a given computation cost. We compare the different types of checksum and give a output to when to use a given checksum.

0.5 objectives of proposed work

To understand which checksum to use when given a computational cost. Which checksum do we use when some kind of error occured(bit or burst).To use the right checksum for the right problem and for a specified cost.

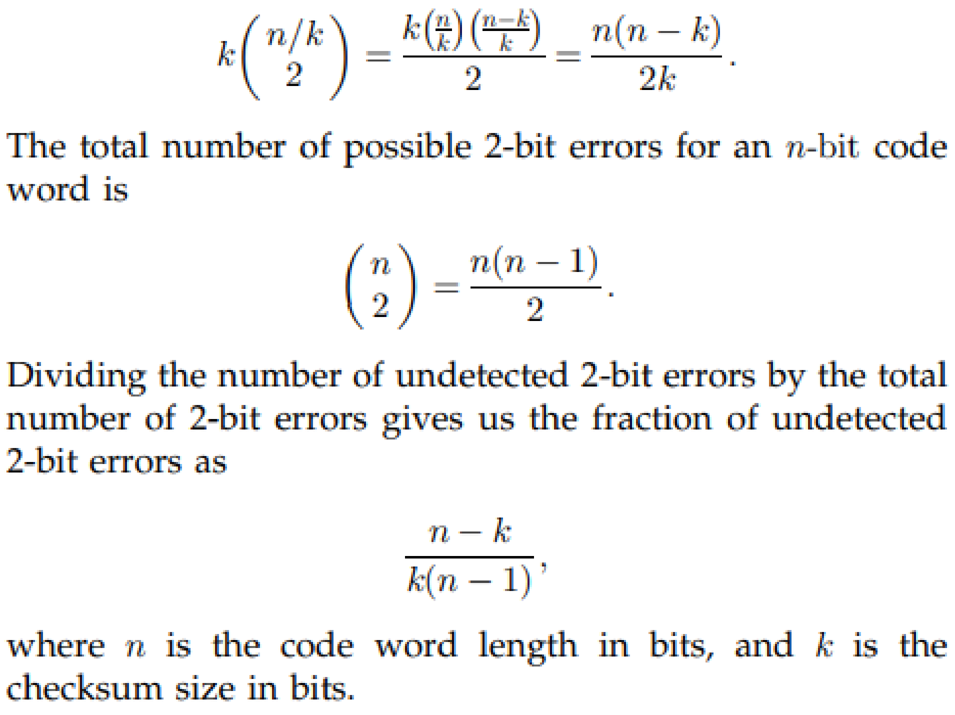
organisation of rest of report:In the literature survey part, we are going to discuss in detail all checksum methods and their undetected probability and in the proposed methodology we will discuss which checksum can be preferred according to the requirements

LITERATURE REVIEW

Data word of length n is divided into k sized blocks then computation is made between them to get checksum which will be attached to dataword to generate codeword as shown in fig1.

xor checksum

Blocks of data word are xored to compute checksum , and order in which blocks are processed is not relevant doesn't affect checksum , bit i of checksum is computed by xoring bits of blocks in ith bitposition , it has hd=2,it can't detect even no of errors occuring in same bitposition , it can detect burst errors greater than k if there are odd error bits in same bit position , it's probability [4]of undetected error is given by, To go undetected any 2 bit of block should have error in same bit position , so we need to choose 2 out of n/k and mutiply by k



$$k\binom{n/k}{2} = \frac{k\binom{\frac{n}{k}}{}\left(\frac{n-k}{k}\right)}{2} = \frac{n(n-k)}{2k}.$$

The total number of possible 2-bit errors for an n-bit code word is

$$\binom{n}{2} = \frac{n(n-1)}{2}.$$

Dividing the number of undetected 2-bit errors by the total number of 2-bit errors gives us the fraction of undetected 2-bit errors as

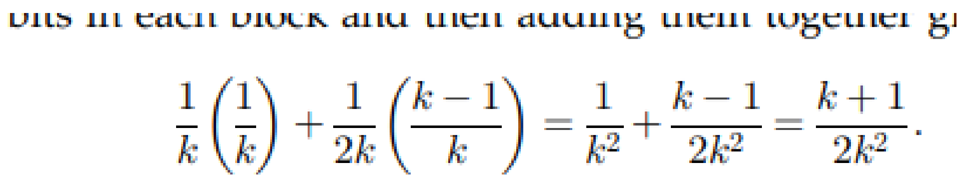
$$\frac{n-k}{k(n-1)},$$

where n is the code word length in bits, and k is the checksum size in bits.

Figure 2: xor pud

2’s complement

Blocks of data word are 2’s complemented to compute checksum , and order in which blocks are processed is not relevant doesn’t affect checksum , carry we get is discarded ,order doesn’t matter it has hd=2 ,it has better detection efficiency than xor due to mixing of bit positions because of bitbybit carry It can’t detect error when there are 2 errors which occurs in same bitposition of different blocks where 1 bit is 1,other is 0 the resultant will give same sum no matter of error And also when any of msb bit is corrupted ,sum remains same carry is discarded It’s probability of undetected error is given by , where msb is corrupted and due to 1st case we get as shown in fig3



bits in each block and then adding them together gi

$$\frac{1}{k}\left(\frac{1}{k}\right) + \frac{1}{2k}\left(\frac{k-1}{k}\right) = \frac{1}{k^2} + \frac{k-1}{2k^2} = \frac{k+1}{2k^2}.$$

Figure 3: 2’s complement pud

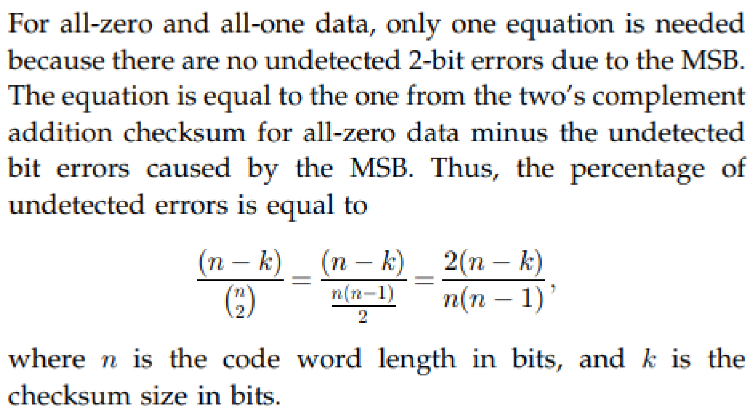
one’s complement

one’s complement checksum method is almost to that of 2’s complement , where checksum is computed by one’s complement on all blocks of data word ,and the carry obtained is added back to lsb therefore it can detect case2 of 2’s complement ,where it can detect corrupted msb , it can detect upto burst erros of k-1 size in length ,other than that it is almost similar to 2’s complement

it’s pud probability of undetected error is given by fig4.

fletcher checksum

Fletcher checksum is more efficient than xor,1’s,2’s checksum method , it’s checksum is computed in such a way where we have block of size half of checksum size where after calculating suma and sumb shown in fig after all blocks they are concatenated to get a checksum ,where d are blocks , the order in which blocks are processed is significant ,it can detect burst error upto half of checksum size ,for a certain modulo code it has hd=3, and for

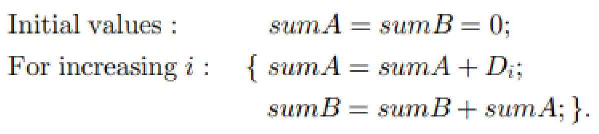

For all-zero and all-one data, only one equation is needed because there are no undetected 2-bit errors due to the MSB. The equation is equal to the one from the two's complement addition checksum for all-zero data minus the undetected bit errors caused by the MSB. Thus, the percentage of undetected errors is equal to

$$\frac{(n-k)}{\binom{n}{2}} = \frac{(n-k)}{\frac{n(n-1)}{2}} = \frac{2(n-k)}{n(n-1)},$$

where n is the code word length in bits, and k is the checksum size in bits.

Figure 4: one's complement pud

all rest of code word length it has hd=2,it can detect 2bit errors of $(2^(k/2)-1)*k/2$ sized or less than data word and modulo is $2^(k/2)-1$



$$\begin{array}{ll} \text{Initial values :} & sumA = sumB = 0; \\ \text{For increasing } i : & \{\ sumA = sumA + D_i; \\ & sumB = sumB + sumA; \}. \end{array}$$

Figure 5: fletcher checksum

adler checksum

Adler checksum is almost similar to fletcher checksum , but while computing we used$2^k/2-1$ as modulo but here we use prime modulo and computing algorithm is almost similar where we initially set suma=1 and modulo is 65521 for 32bit checksum , it results in better mixing of bits ,it can detect 2 bit error for m*k/2 sized dataword m refers to modulo k to size of checksum, for greater than this length it can detect 1 error ,and all burst errors for codeword length less than k/2

PROPOSED METHODOLOGY

Checksum effectiveness based on data error and it's distribution
For a k bit checksum there are 2^k possibilities of checksum values hence the probability of detecting it is $1/2^k$ but it is believed that every algorithms have the same value which is a misconception. It may be valid for random data but in reality there are no random data and during corruption it affects few bits

Like in xor checksum method, only 1 bit of FCS is changed if 1 bit is corrupted and even add checksum method also changes 1 bit of FCS whereas fletcher and addler changes several bits likely to affect more in high half , and in case of crc every bit may be affected

2's compliment.1's compliment ,fletcher,addler are data dependent checksum methods , so pud is affected by the data content whereas xor,crc checksum methods are data independent ,pud will be less when there are all o's are all 1's ,but will be high if there are more equal no of 1 and 0 in bit position

Fletcher Checksum Compared to Cyclic Redundancy Codes

The fletcher checksum has capacity to detect error of$1/(2^(k/2)-1\ )$ which is less compared to $1/2^k$ regardless of their burst error and length , if k is checksum size ,in most of the embedded systems the code word length will be less than2^k , for ex Ethernet Maximum Transmission Unit (MTU) is only 12,000 bits, which is less than 65,535 bits in these scenarios CRC is better than fletcher

One’s Complement Fletcher Checksum Compared to the Adler Checksum

It is known that adler checksum has better error detective capacity than fletcher because of prime modulus by which bits are mixed up , but there are only few valid FCS for codewords thus this a negative aspect of it inspite of better mixing up of bits thus fletcher is more superior compared to fletcher ,but for less sized data word adler 16 is good ,but for modular addition ther’s an increase in complexity and computational cost.

ERROR DETECTION AND COMPUTATIONAL Cost hindrance

Cost Performance hindrance

Computational cost of one’s compliment and fletcher is less than CRC , where CRC uses mathematical operations as polynomial division which is more complex , in order to select a checksum for a given network there are lot of factors such as compatibility, required level of error detection , some prefer less complex computation over error detection efficiency others may prefer accurate error detection effiency over high computation Among all xor checksum has less computation but less error detection efficiency and does not vary much with codeword length , and 2’s compliment addition checksum has similar performance but it’s error detection is affected by codeword length .

1’s compliment addition checksum has almost or a bit higher computational cost than add checksum 2’s compliment , carry bit of MSB is incorpated and added to LSB ,thus compared to xor,add , 1’s compliment has a bit more computational cost ,but offers better error detectiom and is not much affected by cord word length , therefore 1’s compliment is more preferred over xor and add .

Fletcher checksum has a higher computational cost than one’s complement ,because it has 2 sum variable , but gives a better error detection for cord word of long lengths , it’s HD=3 for short code word lengths, thus has good performance than 1’s complement ,but performance gets affected when it goes beyond HD=3 .

We know that adler checksum has higher computational cost than that of fletcher because of prime modulus ,but similiarly after a certain code word length it’s effectiveness decreases . Fletcher checksum is preferred for code word length less than2^k ,but CRC is preffered for longer code word length which is 10 times better than fletcher , but CRC performance varies lot as per code word length compared to others Conclusion to choose checksum

Use 2’s complement over xor checksum which has better error detection , similiarly it Is preferable to use 1’s complement over 2’s when we deal with random independent bit errors And 2’s complement is preferred over 1’s when there are burst errors , but whenever there is both burst and random independent 1’s complement is more preferred If high computational cost is affordable then fletcher is more prefferd over 1’s,2’s, for random independent bit error but even fletcher is not advisable for more burst errors and dataword consisting of more zeroes or ones

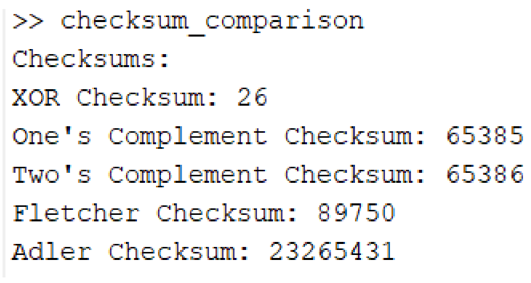
Implementation

we can study the various mentioned checksum algorithms through the matlab code

```
function checksum_comparison()
    % XOR Checksum
    function xorChecksum = calculateXORChecksum(data)
        xorChecksum = bitxor(data(1), data(2));
        for i = 3:length(data)
            xorChecksum = bitxor(xorChecksum, data(i));
        end
    end

    % One's Complement Checksum
    function onesComplementChecksum = calculateOnesComplementChecksum(data)
        onesComplementChecksum = sum(data);
        while onesComplementChecksum > 65535
            onesComplementChecksum = mod(onesComplementChecksum, 65536) + floor(onesCompler
        end
        onesComplementChecksum = bitcmp(uint16(onesComplementChecksum));
    end
    % Two's Complement Checksum
    function twosComplementChecksum = calculateTwosComplementChecksum(data)
        twosComplementChecksum = sum(data);
        while twosComplementChecksum > 65535
            twosComplementChecksum = mod(twosComplementChecksum, 65536) + floor(twosComple
        end
        twosComplementChecksum = bitcmp(uint16(twosComplementChecksum)) + 1;
    end

    % Fletcher Checksum
    function fletcherChecksum = calculateFletcherChecksum(data)
        sum1 = 0;
        sum2 = 0;
        for i = 1:length(data)
            sum1 = mod(sum1 + data(i), 65535);
            sum2 = mod(sum2 + sum1, 65535);
        end
        fletcherChecksum = bitshift(sum2, 8) + sum1;
    end

    % Adler Checksum
    function adlerChecksum = calculateAdlerChecksum(data)
        prime = 65521;
        sum1 = 1;
        sum2 = 0;
        for i = 1:length(data)
            sum1 = mod(sum1 + data(i), prime);
            sum2 = mod(sum2 + sum1, prime);
        end
        adlerChecksum = bitshift(sum2, 16) + sum1;
    end

    % Test data
    data = [10 20 30 40 50];

    % Calculate and display checksums
    xorChecksum = calculateXORChecksum(data);
    onesComplementChecksum = calculateOnesComplementChecksum(data);
    twosComplementChecksum = calculateTwosComplementChecksum(data);
    fletcherChecksum = calculateFletcherChecksum(data);
    adlerChecksum = calculateAdlerChecksum(data);

    % Display checksums
    disp('Checksums:');
    disp(['XOR Checksum: ' num2str(xorChecksum)]);
    disp(['One''s Complement Checksum: ' num2str(onesComplementChecksum)]);
    disp(['Two''s Complement Checksum: ' num2str(twosComplementChecksum)]);
    disp(['Fletcher Checksum: ' num2str(fletcherChecksum)]);
    disp(['Adler Checksum: ' num2str(adlerChecksum)]);
end
```

Results

Here fig6 represents the output of the Matlab code and
fig7 shows the graph of the probability of undetected errors vs codeword length of random independent errors



```
>> checksum_comparison
Checksums:
XOR Checksum: 26
One's Complement Checksum: 65385
Two's Complement Checksum: 65386
Fletcher Checksum: 89750
Adler Checksum: 23265431
```

Figure 6: output

We can observe from the figure7,that as the probability of undetected increases the efficiency of that checksum algorithm is not good or not efficient,
We can confirm from the figure that analysis and comparison of checksum algorithms which we did in previous sections is true , where efficiency of checksum algorithms for random independent bit errors increases in the order of xor,2's complement addition,1's complement addition,fletcher ,adler.

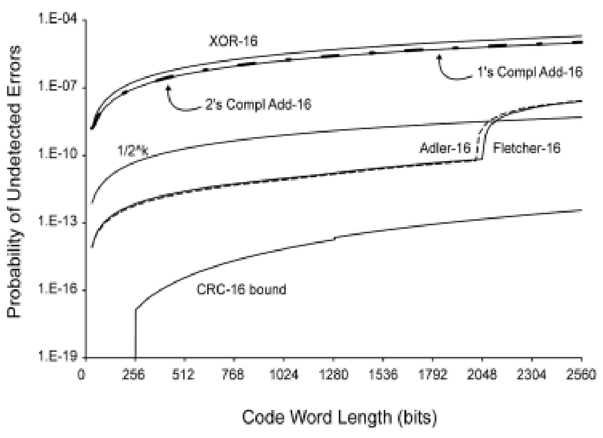


Figure 7:

Conclusions

From the analysis and brief study and comparison of different checksum algorithms we can conclude that as a general notion, it is false that in real networks the probability of undetected is always $1/2^k$ instead it depends on other factors as well such as codeword length, the algorithm used, type of data and type of error. performance of algorithms is actually better than $1/2^k$.
If there's a constraint on computational cost,we can go with Fletcher checksum rather than crc which is used widely nowadays. and has less computational cost than Adler and in most conditions offer better performance than adler.
and even if there's more computational cost constraint, then we can go with one's complement addition checksum which outperforms in the efficiency of 2's complement and has similar computational cost.
but if there's no constraint on computational cost, then rather than these checksum algorithms it is better to go with CRC error detection algorithm.
the choice of checksum algorithm depends on the specific requirements of your application. If you require basic error detection, XOR checksum can be sufficient. However, for stronger error detection, 2's complement addition or 1's complement addition can be considered. If you need more robust error detection capabilities, Fletcher checksum is a good choice, while Adler checksum strikes a balance between error detection and computational efficiency.
Anyone can refer this paper for choosing the correct checksum algorithm based on their requirements,needs and constraints.

References

[1] Error Detection Code – Checksum, https://www.geeksforgeeks.org/error-detection-code-checksum/ (2020).

[2] checksum, https://en.wikipedia.org/wiki/Checksum.

[3] what is checksum, https://www.scaler.com/topics/computer-network/checksum/.

[4] Error masking probability of 1's complement checksums, https://ieeexplore.ieee.org/document/956312 (2001).

[5] Fletcher's checksum, https://en.wikipedia.org/wiki/Fletcher%27s_checksum.

[6] implement checksum, https://www.geeksforgeeks.org/c-c-program-to-implement-checksum/?ref=rp.

[7] The Effectiveness of Checksums for Embedded Control Networks, https://ieeexplore.ieee.org/document/4358707 (2009).

[8] checksum error detection, https://www.researchgate.net/publication/364928338_Checksum_Error_Detection.

[9] adler, https://en.wikipedia.org/wiki/Adler-32.

[10] Double and triple error detecting capability of Internet checksum and estimation of probability of undetectable error, https://ieeexplore.ieee.org/document/640124.

****** END ******
Note: [1–10]