INDRA Note No. 679
24 July, 1978

IEN 49
section # 2.4.4.6

INDRA
Working Paper

COMMENTS ON TCP CHECKSUM FUNCTION DESIGN: A RESPONSE TO INTERNET EXPERIMENT NOTE 45.

by P.L. Higginson

ABSTRACT

This paper comments on the proposed 'ones complement sum' and 'reduced product code' schemes outlined in IEN 45. It is argued that the product code suggested offers no real gains over the ones complement method and that both methods are unsuitable for use with TCP. It is shown that cyclic redundancy code checks take about twice as long to compute as the reduced product code but offer several advantages and furthermore that they can be designed to allow easy updating in the gateways without recalculation.

Dept of Statistics & Computer Science
University College, London

Comments on TCP Checksum Function Design :

A Response to Internet Experiment Note 45

by P.L. Higginson

Abstract

This paper comments on the proposed 'ones complement sum' and 'reduced product code' schemes outlined in IEN 45. It is argued that the product code suggested offers no real gains over the ones complement method and that both methods are unsuitable for use with TCP. It is shown that cyclic redundancy code checks take about twice as long to compute as the reduced product code but offer several advantages and furthermore that they can be designed to allow easy updating in the gateways without recalculation.

1. Introduction

Internet Experiment Note 45 (IEN 45) discusses several checking schemes and finally seems to favour a 'reduced product code' scheme. While I have no experience of product codes in general, the reduced one proposed is described in sufficient detail in IEN 45 for the purposes here. However, it is clear that the comments I have do not apply to other product codes which might be used. The following sections describe my major problems with the proposal scheme and then go on to detail how a cyclic redundancy code check of the systematic type (i.e. used for error detection only) might be used and how it could be updated by gateways. Finally, some minor comments on IEN 45 are made.

2. Objects of a TCP Checksum

The TCP checksum is a second level check done on an end-to-end basis. It is assumed that error prone mediums such as Radio, Telephone and Satellite channels will have low level CRC checks implemented hop-by-hop. The object of the TCP checksum is to detect such errors as memory failures, interface and software errors in the network(s), and local interface errors as well as gross distortions which slip through the hop-by-hop CRC's.

Given that the data is in many places passing in parallel through registers and interfaces, property 1 (Section 2 of IEN 45) by which a check is commutative is most undesirable since it means that two errors in the same bit position can compensate each other in the checksum. So P1 implies that only 1 error can be detected whereas a normal CRC check will detect 1,2 or 3 errors in a message.

3. Problems with the Reduced Product Code

Since the checking method for the reduced product code is identical to the ones complement method it follows that it detects and fails to detect exactly the same cases. Thus it offers no checking advantage over ones complement and in fact if two compensating errors in words n and m do occur, the result when the message is decoded will be to corrupt all words between and including n and m instead of only n and m.

4. Calculating a CRC Check

The standard table look up method for calculating any CRC check requires both table space and calculation time and these can be traded. One possible way on the PDP-11 is to use a 512 word table and take 15 memory cycles per 16 bit word checked. In general, if you decrease the table size you increase the time taken. Because the CRC check is calculated bit serially it can be done in any byte size (with suitable end correction if necessary); thus a PDP-10 with 36 bit words could use 9 bit bytes and a 512 entry table (the PDP-11 method above used 2 256 word tables - the PDP-10 could use 1,2 or 4 512 by 16 bit tables).

Some CRC polynomials, particularly the IBM one ($X^{16}+X^{15}+X^2+1$) are susceptible to fast coding methods and in (8) we gave a method which on a PDP-9 takes 75% of the time of the standard table lookup using two 256 entry tables. On a PDP-11 we can do this in 13 memory cycles per 16 bit byte.

5. Updating a CRC checksum

CRC checks have a property similar to the algebraic a(b+c)=ab+ac in that

$$r(a \text{ XOR } b) = r(a) \text{ XOR } r(b) \quad .$$

Thus in principle the checksum can be updated by calculating the XOR (exclusive or) of the old and new of the changed part and working out what to XOR into the existing checksum (the length of the message must be known but not its contents). This obeys the requirement that the checksum is updated not recalculated but would be tedious unless the data part was either of short or fixed length (when the tables could be calculated in advance).

The best solution is to calculate the initial CRC check backwards so that the header is added in last and thus can be changed easily. If it was wished to have a forward check then the inverse polynomial ($X^{16}+X^{14}+X+1$ for the IBM case) could be used to calculate the initial backwards checksum and to do the updating. In fact, the checksum could be fitted into the correct position in the header by going both forwards and backwards and XORing the results.

I feel these extra sophistications are unnecessary and the backwards check with the result put into the CRC field is sufficient. The complexity of this is about the same as updating the product case; the number of CRC steps is fixed and the data message unaltered. If a word is inserted, the whole of the shifted data has to be regarded as altered, but this is only the header in the case under consideration (it is XOR'd in twice in its old and new positions).

6. Minor Comments on IEN 45

Section 1, paragraph 2: do you mean upper or lower bound - I think 2**(-C) is a mean rather than a bound anyway.

paragraph 3: all zero messages are now defeated by starting and finishing CRC checks with non-zero values (see X25).

Section 2, P2 : I do not think is necessary because P1 is sufficient.

P4 : P1 gives this result not associativity (+ is associative though).

Section 2, PDP-10 code : you seem to have only 32 bits in each 36 bit word which means you have preprocessed the data (what about 36 bit FTP's?) unless I misunderstand the code.

Section 3, paragraph 6 : All checks we are considering will detect bursts of 15 or fewer bits in error providing you define a burst in the checking order (which if you move the CRC is not strictly true).

paragraph 7 : 3% does not sound very good compared to an expected 1 in 2^{16}.

References : (8) is a published paper - see below.

7. Conclusions

I feel that the three-fold increase in errors detected by a CRC check justifies the extra computation overhead in calculating and checking. The updating problem can be solved if the check is designed well. The TCP check should be additional to any hop-by-hop checks and should not be amalgamated otherwise the error checking will be compromised. Hence a check polynomial other than one likely to be used at line level should be chosen. The old IBM check is suggested with modifications to have non-zero start and finish values in order to minimise the zero fill problems.

8. References

(8) P.L. Higginson and P.T. Kirstein, 'On the computation of cyclic redundancy checks by program'. Computer Journal, 16, 19-22 (1973).