Edgar H. Sibley
Panel Editor

The tea-leaf reader CRC algorithms are error-detection algorithms that use a look-ahead table to increase execution speed.

THE TEA-LEAF READER ALGORITHM: AN EFFICIENT IMPLEMENTATION OF CRC-16 AND CRC-32

GEORGIA GRIFFITHS and G. CARLYLE STONES

We present a method for computing a CRC (Cyclical Redundancy Check), for either a CRC-16 or CRC-32, that is easy to implement in software and takes much less CPU time than conventional methods. Borrowing from cryptological techniques, we provide evidence that CRC computations can be more efficiently handled by a bytewise method other than the standard bitwise approach (up to 19 times faster!).

The CRC algorithms are an error-detecting checksum technique for data packages, which yield a low probability of undetected errors. This undetected-error rate may be as low as 1 undetected error in 10 to the 17th bit. We assume that the reader has a working knowledge of the CRC algorithms. For an explanation of the CRC algorithms and their maximally generating primitive polynomial lists, trade-offs, and error-detection probability predictions, see [1] and [2].

Our method is mathematically equivalent to the existing methods and can therefore be implemented on any computer with any existing interface currently using a CRC. There are personal computers using the CRC that are CPU bound during I/O, which must therefore slow down the BAUD rate of the interface. This algorithm will help to eliminate the CPU timing problems associated with I/O.

The *tea-leaf reader* CRC algorithm is so named because it uses a look-ahead table to determine the effect of the next 8 bits on the CRC value. The trade-off is between memory space and execution speed. The tea-leaf reader method performs calculations "off-line" and stores them in a table, whereas the bitwise algorithm performs calculations upon each input bit. The implementation of the algorithm actually requires the development of two software programs: the first an on-line CRC table lookup algorithm, and the second an off-line table builder. We take a close look at both programs in the colored boxes starting on the next page.

The on-line CRC-32 table lookup algorithm uses five 256-byte tables to calculate the next 8 CRC bits. The 4 parity bytes are used as indexes into the four lookup tables. The four results are exclusive or'ed (XOR'ed) together. This result is XOR'ed with the next input byte, and the result is used as an index into the tea-leaf table to create the next high-order CRC byte (see Figure 1, next page). This algorithm requires only five table look-ups, five XOR's, and four shifts per input byte. For a CRC-16, there are only two parity byte lookups instead of the four for the CRC-32.

The off-line CRC-32 table-builder algorithm determines how the taps (terms of the polynomial) will affect the next eight bit shifts through the CRC parity. The outputs of the table builder are five assembly-language tables that are assembled and linked with the on-line CRC-32 algorithm. For a CRC-16 algorithm, only three tables are needed: two CRC-16 parity byte lookup tables and the next input byte tea-leaf table. To explain the table-builder program, an example of a CRC-16 is shown in the first box (next page).

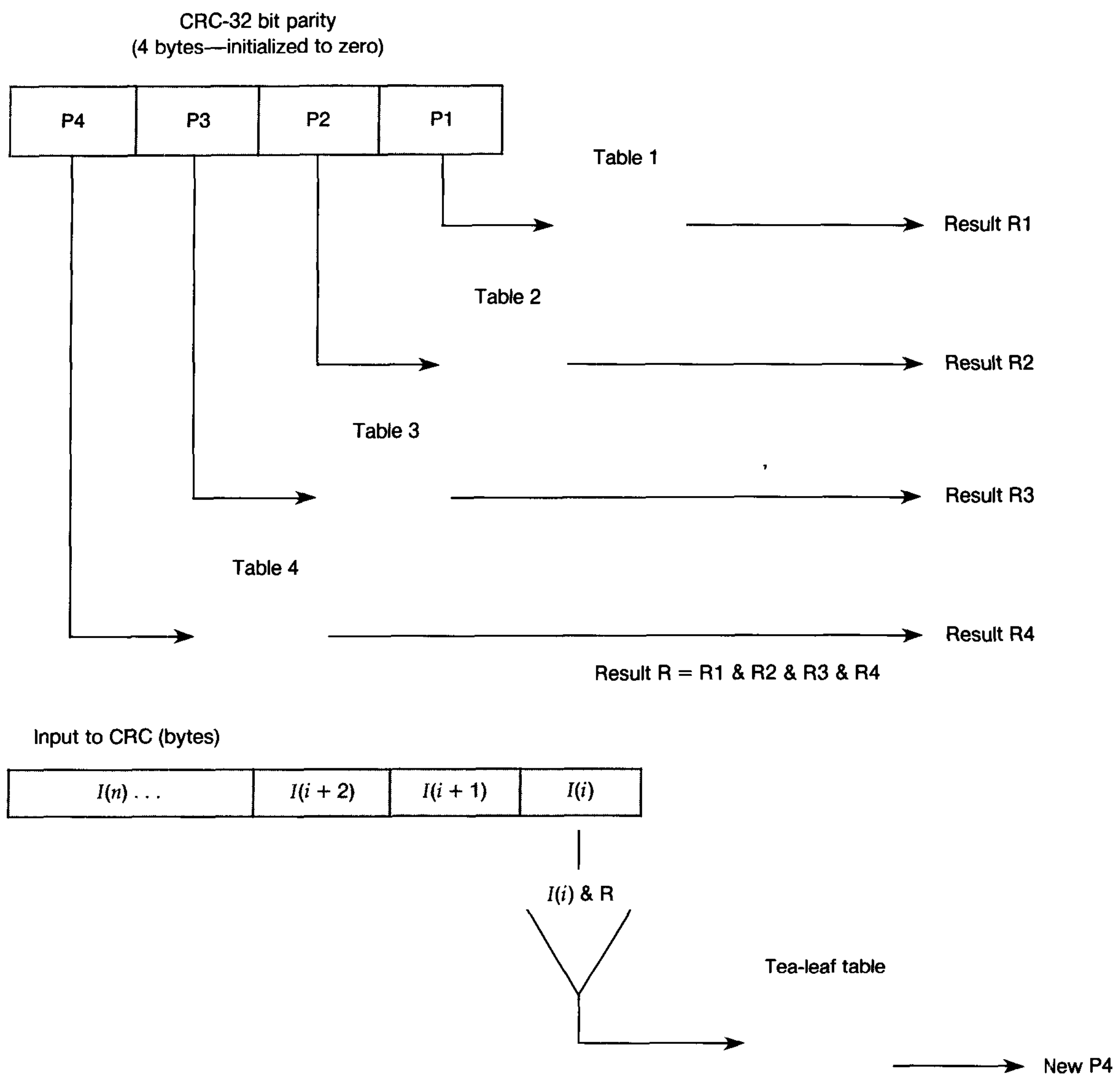


FIGURE 1. (& = XOR function)

CRC-16 Example (& = XOR function)

First the conventional bitwise method of calculating the CRC-16 is presented, and then the tea-leaf reader method. This example demonstrates by construction the equivalency of the two algorithms.

The polynomial to be used is as follows:

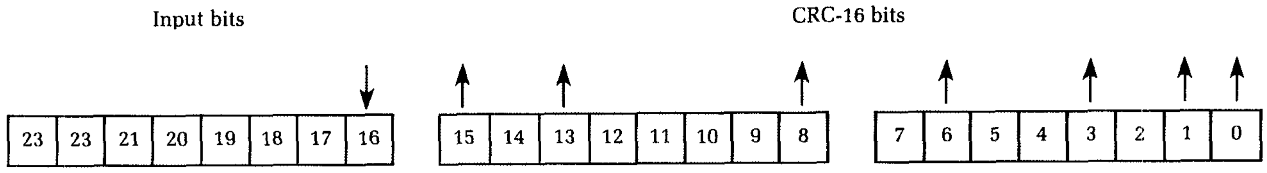
CRC-16 polynomial:
$X^{16} + X^{15} + X^{13} + X^8 + X^6 + X^3 + X + 1.$

The conventional method is a *bit-by-bit algorithm*:

For first bit of INPUT
16 & 15 & 13 & 8 & 6 & 3 & 1 & 0

Shift right into bit 15.

This requires eight XORs for each input bit.



Now for the tea-leaf reader algorithm:

Visualize the & relation being shifted left instead of the bits being shifted right; thus

for first bit	16 & 15 & 13 & 8 & 6 & 3 & 1 & 0	#1
for next bit	17 & #1 & 14 & 9 & 7 & 4 & 2 & 1	#2
	18 & #2 & 15 & 10 & 8 & 5 & 3 & 2	#3
	19 & #3 & #1 & 11 & 9 & 6 & 4 & 3	#4
	20 & #4 & #2 & 12 & 10 & 7 & 5 & 4	#5
	21 & #5 & #3 & 13 & 11 & 8 & 6 & 5	#6
	22 & #6 & #4 & 14 & 12 & 9 & 7 & 6	#7
until finally	23 & #7 & #5 & 15 & 13 & 10 & 8 & 7	#8

To implement this with tables, there must be three tables: two parity lookup tables and the "tea-leaf" table. The input to the first table is the 0–7 byte of the CRC-16 register (the bit pattern in the rightmost eight positions). The output of this first table will be determined by relations #1–#8.

Result R1 (8 bits):

H	G	F	E	D	C	B	A

Since the input consists of bits 0–7, we obtain an output bit in position "A" by examining the significant bits of relation #1.

Thus	"A" =	6 & 3 & 1 & 0	by #1
similarly	"B" =	7 & 4 & 2 & 1	by #2
	"C" =	5 & 3 & 2	by #3
	"D" =	6 & 4 & 3	by #4
	"E" =	7 & 5 & 4	by #5
	"F" =	6 & 5	by #6
	"G" =	7 & 6	by #7
until finally	"H" =	7	by #8

The information content decreases as the relations progress to higher order bits that are not available in the 0–7-bit input.

For the second table, we have as input the byte containing 8–15 bits. Again the input is determined by the relations #1–#8.

Result R2 (8 bits)

P	O	N	M	L	K	J	I

As in the first table, we again examine the significant bits available for relations #1–#8.

"I" =	15 & 13 & 8	by #1
"J" =	14 & 9	by #2
"K" =	15 & 10 & 8	by #3
"L" =	11 & 9	by #4
"M" =	12 & 10	by #5
"N" =	13 & 11 & 8	by #6
"O" =	14 & 12 & 9	by #7
"P" =	15 & 13 & 10 & 8	by #8

As in the first table, the information content decreases as the relation progresses to higher order bits that are not available in the 8–15-bit input, but increases as the lower order bits become available in the progression of the relations.

By XORing the outputs of the first two tables bit by bit, we have a resultant byte R (bits R0–R7).

R(0) = A & I	= (15 & 13 & 8)	& (6 & 3 & 1 & 0)
R(1) = B & J	= (14 & 9)	& (7 & 4 & 2 & 1)
R(2) = C & K	= (15 & 10 & 8)	& (5 & 3 & 2)
R(3) = D & L	= (11 & 9)	& (6 & 4 & 3)
R(4) = E & M	= (12 & 10)	& (7 & 5 & 4)
R(5) = F & N	= (13 & 11 & 8)	& (6 & 5)
R(6) = G & O	= (14 & 12 & 9)	& (7 & 6)
R(7) = H & P	= (15 & 13 & 10 & 8)	& (7)

XORing the resultant byte with the input byte (16–23) gives us an input byte to the "tea-leaf" table (bits T0–T7) at the bottom of this page.

With T0–T7 as input, the remaining task for the tea-leaf table is to complete relations #R1–#R8, but #1 is already complete in T0; thus

Result #2 (8 bits)

H′	G′	F′	E′	D′	C′	B′	A′

A′ = T0	= #1 = new bit 16
B′ = T1 & A′	= #2 = new bit 17
C′ = T2 & B′	= #3 = new bit 18
D′ = T3 & C′ & A′	= #4 = new bit 19
E′ = T4 & D′ & B′	= #5 = new bit 20
F′ = T5 & E′ & C′	= #6 = new bit 21
G′ = T6 & F′ & D′	= #7 = new bit 22
H′ = T7 & G′ & E′	= #8 = new bit 23

Now a right byte shift will cause the following:

8 → 0	and	A′ → 8	and next message byte → 23 ··· 16
9 → 1	and	B′ → 9	
10 → 2	and	C′ → 10	
11 → 3	and	D′ → 11	
12 → 4	and	E′ → 12	
13 → 5	and	F′ → 13	
14 → 6	and	G′ → 14	
15 → 7	and	H′ → 15	

And that's it!

Table bits	Relational equivalences	REL—missing new bits
T0 = R(0) & 16	= 16 & 15 & 13 & 8 & 6 & 3 & 1 & 0	= #1
T1 = R(1) & 17	= 17 & 14 & 9 & 7 & 4 & 2 & 1	= #2 − bit #1
T2 = R(2) & 18	= 18 & 15 & 10 & 8 & 5 & 3 & 2	= #3 − bit #2
T3 = R(3) & 19	= 19 & 11 & 9 & 6 & 4 & 3	= #4 − bits (#3 & #1)
T4 = R(4) & 20	= 20 & 12 & 10 & 7 & 5 & 4	= #5 − bits (#4 & #2)
T5 = R(5) & 21	= 21 & 13 & 11 & 8 & 6 & 5	= #6 − bits (#5 & #3)
T6 = R(6) & 22	= 22 & 14 & 12 & 9 & 7 & 6	= #7 − bits (#6 & #4)
T7 = R(7) & 23	= 23 & 15 & 13 & 10 & 8 & 7	= #8 − bits (#7 & #5)

TABLE I. Timing Comparison

Number of taps and CRC result length	Microseconds per byte for bit method	Microseconds per byte for byte method	Milliseconds for 400-character bit method	Milliseconds for 400-character byte method
17 taps CRC-32	1312	68	524	27
15 taps CRC-32	1184	66	473	27
11 taps CRC-32	928	68	371	27
7 taps CRC-16	608	44	243	17
5 taps CRC-16	480	44	192	17

As the example shows, the CRC algorithm executed without a table lookup is extremely time consuming. For a CRC-32 with a 17-tap polynomial, there are 17 extractions, 17 XORs, and five shifts for each bit of input. This is 312 instructions per byte. With the table lookup method, there are no extractions, five table lookups, 5 XORs, and five shifts, which is 15 instructions per byte. The looping (cycling through the data input) also takes more instructions for the bit method versus the byte method. The bit method requires at least two instructions per bit; the byte method, two instructions per byte. Table I shows the timing tradeoffs between the two methods. The timing estimate assumes an assembly-language address accessing instruction cycle time of 4 microseconds. The comparison also assumes the availability of an XOR and masking instructions, and no clever register manipulation that could improve CPU time.

In conclusion, the CRC-32 algorithm can help to solve I/O throughput problems. Previous design tradeoffs were between the accuracy of the error-detection algorithm and the execution speed of the I/O routine. If the CPU was too slow or the I/O transmission speed too fast, a weaker, more expedient error-detection routine (CRC-16, LRC, or checksum) would be implemented. With the tea-leaf reader algorithm, this error-detection strength–speed penalty will no longer be a limitation to system designs. By using the CRC-32, CPU utilization will increase, thereby improving the computer system, and possibly the network, performance.

REFERENCES

1. Martin, J. *Security, Accuracy, and Privacy in Computer Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1973.
2. Peterson and Weldon. *Error Correcting Codes.* MIT Press, Cambridge, Mass., 1972.

CR Categories and Subject Descriptors: D.1.m [**Programming Techniques**]: Miscellaneous; D.2.3 [**Software Engineering**]: Coding; D.2.m [**Software Engineering**]: Miscellaneous
General Terms: Algorithms, Reliability
Additional Key Words and Phrases: CRC algorithms, tea-leaf reader algorithm

Authors' Present Address: Georgia Griffiths and G. Carlyle Stones, Compusec, Inc., 5333 Mission Center Road, Suite 100, San Diego, CA 92108-1302.