THE TCP/IP CHECKSUM

William Stallings

The error-detection calculation used by IP and TCP is based on ones-complement addition. This concept is explained first, followed by a definition of the error-detection calculation.

Ones-Complement Addition

Ones-complement addition is a calculation performed on binary integers. Before defining the addition algorithm, we first look at the way in which integers may be represented in binary form.

There are several alternative conventions used to represent negative as well as positive integers, all of which involve treating the most significant (leftmost) bit in the word as a sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative.

The simplest form of representation that employs a sign bit is the **sign-magnitude representation**. In an *n*-bit word, the rightmost *n* – 1 bits hold the magnitude of the integer.

```
+18  =  00010010
–18  =  10010010      (sign magnitude)
```

So, an 8-bit word can represent values in the range –127 to +127.

With sign-magnitude representation, there are two representations for zero:

$$+0_{10} = 00000000$$
$$-0_{10} = 10000000 \quad \text{(sign magnitude)}$$

This is inconvenient, because it is slightly more difficult to test for 0 (an operation performed frequently on computers) than if there were a single representation.

Another drawback to sign-magnitude representation is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.

Like sign magnitude, **ones complement representation** uses the most significant bit as a sign bit, making it easy to test whether an integer is positive or negative. It differs from the sign-

magnitude representation in the way that the other bits are interpreted, which leads to simpler algorithms for addition and subtraction.

We need to distinguish between an operation and a representation. To perform the **ones-complement operation** on a set of binary digits, replace 0 digits with 1 digits and 1 digits with 0 digits.

X	= 01010001
ones-complement of X	= 10101110
Y	= 10101110
ones-complement of Y	= 01010001

Note that the ones-complement of the ones-complement of a number is the original number.

The ones-complement representation of binary integers is defined as followed. Positive integers are represented in the same way as in sign-magnitude representation. A negative integer is represented by the ones-complement of the positive integer with the same magnitude.

+18	= 00010010
–18 = ones-complement of +18	= 11101101

Note that because all positive integers in this representation have the left-most bit equal to 0, all negative integers necessarily have the leftmost bit equal to 1. Thus the leftmost bit continues to function as a sign bit.

In ordinary arithmetic, the negative of the negative of a number gives you back that number. This is also true in ones-complement arithmetic.

–18	= 11101101
+18 = ones-complement of –18	= 00010010

As with sign-magnitude, ones-complement has two representations of zero:

$+0_{10}$ = 00000000
-0_{10} = 11111111 (ones-complement)

We can now turn to a consideration of ones-complement addition. It should be intuitively obvious that the simplest implementation of addition for signed binary integers is one in which the numbers can be treated as unsigned integers for purposes of addition. This approach does not work for the sign-magnitude representation. For example, these are clearly incorrect:

```
  0011 = +3                      0001 = +1
+ 1011 = –3                    + 1110 = –6
  1110 = –6 (sign-magnitude)     1111 = –7 (sign-magnitude)
```

For sign-magnitude numbers, correct addition and subtraction involve the comparison of signs and relative magnitudes of the two numbers.

With ones-complement addition, however, the straightforward approach, with a minor refinement, works:

```
  0011 = +3                      0001 = +1
+ 1100 = –3                    + 1001 = –6
  1111 =  0 (ones-complement)    1010 = –5 (ones-complement)
```

This scheme will not always work unless an additional rule is added. If there is a carry out of the leftmost bit, add 1 to the sum. This is called an end-around carry.

```
  1101 = –2                      0111 = +7
+ 1011 = –4                    + 1100 = –3
 11000                          10011
   ___1                           ___1
  1001 = –6 (ones-complement)    0100 = +4 (ones-complement)
```

Application to IP and TCP

For the IP error detection operation, the entire header of an IP datagram is treated as a block of 16-bit binary integers in ones-complement representation. To compute the checksum, the

checksum field in the header is first set to all zeros. The checksum is then calculated by performing ones-complement addition of all the words in the header, and then taking the ones-complement operation of the result.

The identical computation is performed for TCP. In this case, the computation is performed on the words comprising the segment header, the segment data, plus a pseudoheader that includes the following fields from the IP header: source address, destination address, TCP's protocol identifier, and the length of the TCP segment. If the segment contains an odd number of octets, the last octet is padded out on the right with zeros to form a 16-bit word. As with the IP algorithm, the checksum field is set to zero for the calculation.