

TCP-IP Checksums

Alex Urich

Internet Checksum

IP Checksum Definition

The IP checksum is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. One question many people may ask is "What is the 1's complement sum?". This is because all computers utilize the 2's complement representation and the 1's complement is not used.

The following gives a short introduction.

2's complement fixed point integers (8-bit)

Binary	Decimal	Hex
0000 0000	0	00
0000 0001	1	01
0000 0010	2	02
0000 0011	3	03
1111 1111	-1	FF
1111 1110	-2	FE
1111 1101	-3	FD

Let's add two integers:

```
-3 + 5 = 2
FD + 05 = 01 02
```

Discarding the carry (01) gives the correct result.

1's complement fixed point integers (8-bit)

Binary	Decimal	Hex
0000 0000	0	00
0000 0001	1	01
0000 0010	2	02
0000 0011	3	03
1111 1111	-0	FF
1111 1110	-1	FE
1111 1101	-2	FD

IP Checksum Introduction

```
1111 1100 -3 FC
```

Add the same numbers:

```
-3 + 5 = 2
FC + 05 = 01 01
```

Adding the carry (01) to the LSB (01) gives the correct result:

```
01 + 01 = 02
```

So, the 1's complement sum is done by summing the numbers and adding the carry (or carries) to the result..

Simple Internet checksum example

Suppose we have an 8-bit, 2's complement, machine and send the packet

TCP-IP Checksums

Alex Urich

FE 05 00

where 00 is the checksum field.

Let's calculate and verify the Internet checksum.

FE + 05 = 01 03

This is the result of the normal (2's complement) addition. The 1's complement sum requires the addition of the carry to the 8-bit word (even though we will not get the same result)

03 + 01 = 04

so the 1's complement sum of FE + 05 is 04.

The 1's complement of the 1's complement sum (Internet checksum) will be

~04 = FB

and the packet will be sent as

FE 05 FB

Now, at the receiving end we add all the received bytes, including the checksum (again using the 2's complement representation)

FE + 05 + FB = 01 FE

The 1's complement sum is

FE + 01 = FF = -0

which checks that the transmission was OK (see below).

A more complex example (32-bit machine)

As shown in RFC 1071, the checksum calculation is done in the following way:

1. Adjacent octets to be checksummed are paired to form 16-bit integers, and the 1's complement sum of these 16-bit integers is formed.
2. To generate a checksum, the checksum field itself is cleared, the 16-bit 1's complement sum is computed over the octets concerned, and the 1's complement of this sum is placed in the checksum field.
3. To check a checksum, the 1's complement sum is computed over the same set of octets, including the checksum field. If the result is all 1 bits (-0 in 1's complement arithmetic), the check succeeds.

Packet

01 00 F2 03 F4 F5 F6 F7 00 00

(00 00 is the checksum field)

Form the 16-bit words

TCP-IP Checksums

Alex Urich

0100 F203 F4F5 F6F7

Calculate 2's complement sum

0100 + F203 + F4F5 + F6F7 = 0002 DEEF (store the sum in a 32-bit word)

Add the carries (0002) to get the 16-bit 1's complement sum

DEEF + 002 = DEF1

Calculate 1's complement of the 1's complement sum

~DEF1 = 210E

We send the packet including the checksum 21 0E

01 00 F2 03 F4 F5 F6 F7 21 0E

At the receiving

0100 + F203 + F4F5 + F6F7 + 210E = 0002 FFFD

FFFD + 0002 = FFFF

which checks OK.

Comments

It may look awkward to use a 1's complement addition on 2's complement machines. This method however has its own benefits. Probably the most important is that it is endian independent. Little Endian computers store hex numbers with the LSB last (Intel processors for example). Big Endian computers put the LSB first (IBM mainframes for example). When carry is added to the LSB to form the 1's complement sum (see the example) it doesn't matter if we add 03 + 01 or 01 + 03. The result is the same.

Other benefits include the easiness of checking the transmission and the checksum calculation plus a variety of ways to speed up the calculation by updating only IP fields that have changed.

TCP Checksum Calculation

To calculate TCP checksum a "pseudo header" is added to the TCP header. This includes:

- IP Source Address 4 bytes
- IP Destination Address 4 bytes
- TCP Protocol 2 bytes
- TCP Length 2 bytes

The checksum is calculated over all the octets of the pseudo header, TCP header and data. If the data contains an odd number of octets a pad, zero octet is added to the end of data. The pseudo header and the pad are not transmitted with the packet.

In the example code, `u16 buff[]` is an array containing all the octets in the TCP header and data. `u16 len_tcp` is the length (number of octets) of the TCP header and data. `BOOL padding` is 1 if data has an even number of octets and 0 for an odd number. `u16 src_addr[4]` and `u16 dest_addr[4]` are the IP source and destination address octets.

TCP-IP Checksums

Alex Urich

```
/*
*****
Function: tcp_sum_calc()
*****
Description:
Calculate TCP checksum
*****
*/
typedef unsigned short u16;
typedef unsigned long u32;
http://www.netfor2.com/tcpsum.htm (1 of 3)9/16/2004 3:20:20 PM
TCP Checksum Code
u16 tcp_sum_calc(u16 len_tcp, u16 src_addr[],u16 dest_addr[], BOOL
padding, u16 buff[])
{
u16 prot_tcp=6;
u16 padd=0;
u16 word16;
u32 sum;
// Find out if the length of data is even or odd number. If odd,
// add a padding byte = 0 at the end of packet
if (padding&1==1){
padd=1;
buff[len_tcp]=0;
}
//initialize sum to zero
sum=0;
// make 16 bit words out of every two adjacent 8 bit words and
// calculate the sum of all 16 vit words
for (i=0;i<len_tcp+padd;i=i+2){
word16 =((buff[i]<<8)&0xFF00)+(buff[i+1]&0xFF);
sum = sum + (unsigned long)word16;
}
// add the TCP pseudo header which contains:
// the IP source and destination addresses,
for (i=0;i<4;i=i+2){
word16 =((src_addr[i]<<8)&0xFF00)+(src_addr[i+1]&0xFF);
sum=sum+word16;
}
for (i=0;i<4;i=i+2){
word16 =((dest_addr[i]<<8)&0xFF00)+(dest_addr[i+1]&0xFF);
sum=sum+word16;
}
// the protocol number and the length of the TCP packet
sum = sum + prot_tcp + len_tcp;
// keep only the last 16 bits of the 32 bit calculated sum and add
the carries
while (sum>>16)
sum = (sum & 0xFFFF)+(sum >> 16);
// Take the one's complement of sum
sum = ~sum;
return ((unsigned short) sum);
}
```