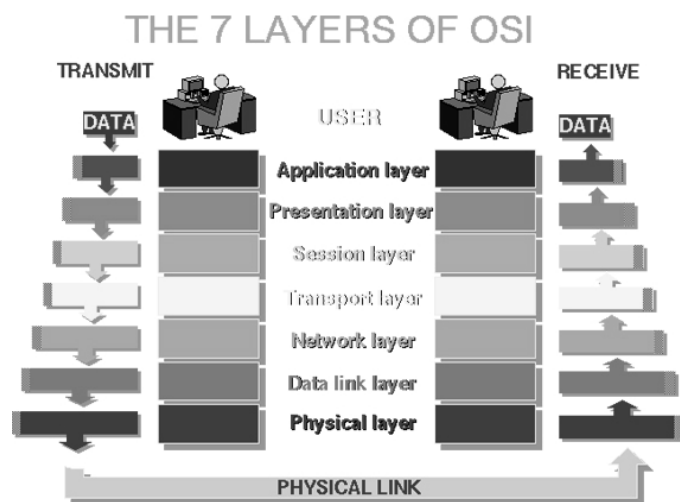


Error detection

David Morgan

© David Morgan 2009

How data gets enveloped



© David Morgan 2009

A trivial example

- I want to send you ABC
 - I send you

A	B	C
---	---	---
- you get ABC
 - you don't know if it's what I sent
 - maybe I sent ABD or something

© David Morgan 2009

A trivial example

- I want to send you ABC
 - A,B,C have ascii codes
65,66,67 totaling 198
 - I send you

A	B	C	¹ 9
			8
- you get ABC or BBC or something
 - you add the ascii codes, compare sums
 - if they disagree it's *not* as I sent it
 - if they agree it's *undetermined* whether it's as I sent it (maybe you get BAC*)

ASCII Alphabet Characters

Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

* two wrongs *do* make a right

Error detection mechanisms

- based on redundancy
 - ultimate redundancy – send an entire 2nd copy
 - effective, too expensive
- lower level protocols
 - VRC vertical redundancy check – parity
 - LRC longitudinal redundancy check – row/column
 - CRC cyclic redundancy check
- higher level protocols
 - checksums

© David Morgan 2009

Parity checking

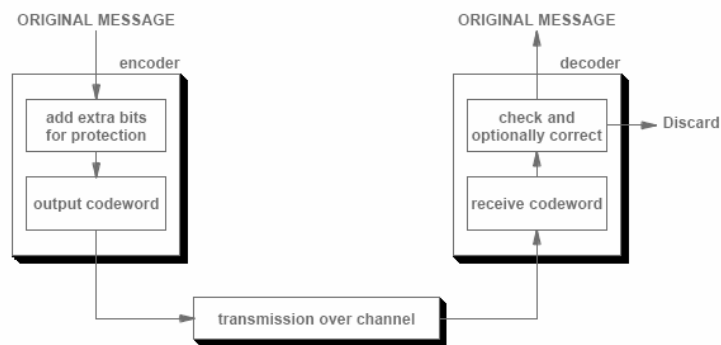


Figure 8.3 The conceptual organization of a forward error correction mechanism.

from our textbook, Computer Networks and Internets, Douglas Comer, Pearson Education Inc, p. 138

© David Morgan 2009

Parity checking

Original Data	Even Parity	Odd Parity
0 0 0 0 0 0 0 0	0	1
0 1 0 1 1 0 1 1	1	0
0 1 0 1 0 1 0 1	0	1
1 1 1 1 1 1 1 1	0	1
1 0 0 0 0 0 0 0	1	0
0 1 0 0 1 0 0 1	1	0

Figure 8.4 Data bytes and the corresponding value of a single parity bit when using even parity or odd parity.

from our textbook, [Computer Networks and Internets](#), Douglas Comer, Pearson Education Inc, p. 139

© David Morgan 2009

An example – even parity

- I want to send you ABC
 - i.e., 01000001 01000010 01000011
 - instead I send 010000010 010000100 010000111
- you get 010000010 011000100 010000111
 - you make sure number of 1-bits always even
 - it isn't (number of 1-bits in middle nonette is odd)
 - data's *not* as I sent it

ASCII Alphabet Characters

Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

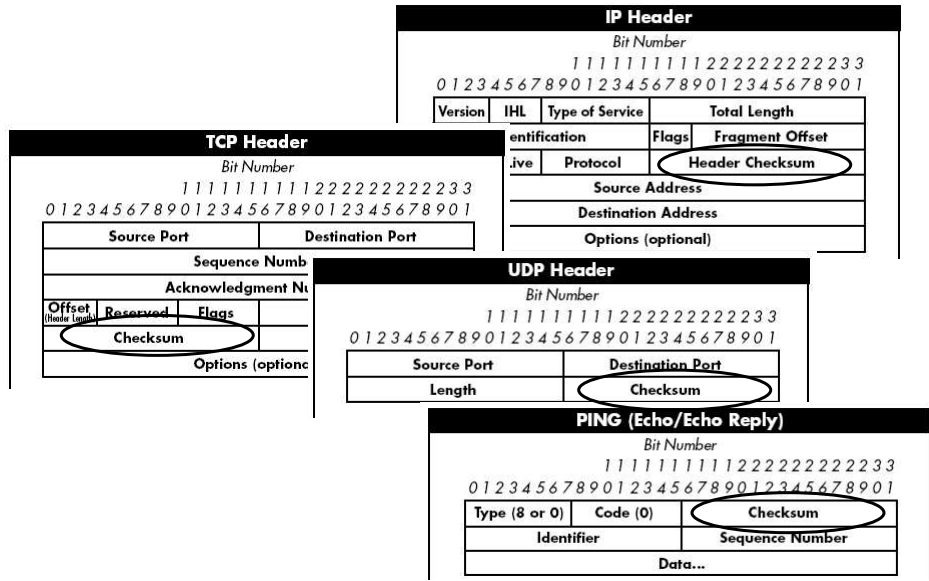
An example – even parity

- I want to send you ABC
 - i.e., 01000001 01000010 01000011
 - instead I send 010000010 010000100 010000111
- you get 010000010 011100100 010000111
 - you make sure number of 1-bits always even
 - it is
 - *undetermined* whether data's as I sent it (in this example particularly, is it or not?)

ASCII Alphabet Characters

Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

Protocols may feature checksums



Checksum? what, exactly?

- what do they sum?
- how do they perform the summation?

© David Morgan 2009

Internet Protocol

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error....

Header Checksum: 16 bits

A checksum on the header only....

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

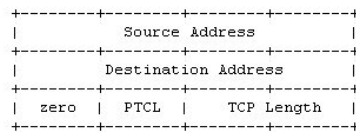
rfc791

© David Morgan 2009

TCP Protocol

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.



The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

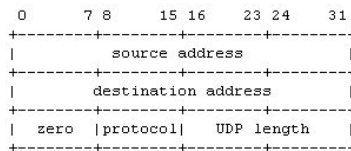
rfc973

© David Morgan 2009

UDP Protocol

...Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as is used in TCP.



If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

rfc768

© David Morgan 2009

ping (ICMP echo/echo reply)

Echo or Echo Reply Message

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|  Type      |  Code      |  Checksum      |
+-----+-----+-----+-----+
|  Identifier  |  Sequence Number  |
+-----+-----+-----+-----+
|  Data ...   |
+-----+
```

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

rfc792

© David Morgan 2009

Checksum? what, exactly?

- *what do they sum?*
- *how do they perform the summation?*

“The checksum is the 16-bit one's complement of the one's complement sum of...

...all 16 bit words in the header.” (IP)

...all 16 bit words in the header and text. (TCP)

...a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. (UDP)

...the ICMP message starting with the ICMP Type. (ping/ICMP echo)

© David Morgan 2009

Checksum? what, exactly?

- what do they sum?
- *how do they perform the summation?*

“The checksum is the 16-bit one's complement of the one's complement sum of...

...all 16 bit words in the header.” (IP)

...all 16 bit words in the header and text. (TCP)

...a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. (UDP)

...the ICMP message starting with the ICMP Type. (ping/ICMP echo)

© David Morgan 2009

What's “one's complement”?

- bitwise inverse
- binary NOT

- one's complement of 0000 is 1111
- one's complement of 1111 is 0000
- one's complement of 1010 is 0101
- one's complement of 1101 is 0010

© David Morgan 2009

What's "one's complement sum"?

- binary sum, plus 1 if a carry/spillover resulted
- 1000 plus 0001 is 1001
- 1000 plus 0111 is 1111
- 1000 plus 1000 is 10000, becomes 0000+1 or 0001
- 1000 plus 1111 is 10111, becomes 0111+1 or 1000
- 1101 plus 1001 is 10110, becomes 0110+1 or 0111

© David Morgan 2009

"one's complement" — again, in hex

- bitwise inverse
- binary NOT
- one's complement of 0 is F
- one's complement of F is 0
- one's complement of A is 5
- one's complement of D is 2
- one's complement of 33E9 is CC16

0011001111101001 1100110000010110

© David Morgan 2009

“one’s complement sum” – again, in hex

- binary sum, plus 1 if a carry/spillover resulted
- 8 plus 1 is 9
- 8 plus 7 is F
- 8 plus 8 is 10, becomes 0+1 or 1
- 8 plus F is 17, becomes 7+1 or 8
- D plus 9 is 16, becomes 6+1 or 7
- C58B plus 82E6 is 14871, becomes 4871+1 or 4872

© David Morgan 2009

Example – IP header checksum

```
1 70 0.000000 130.230.52.139 130.230.52.5 DNS Standard query A www.tut.fi
Frame 1 (70 bytes on wire, 70 bytes captured)
Ethernet II, Src: 00:06:1b:d3:25:19 (00:06:1b:d3:25:19), Dst: 00:14:5e:28:2e:6a (00:14:5e:28:2e:6a)
Internet Protocol, Src: 130.230.52.139 (130.230.52.139), Dst: 130.230.52.5 (130.230.52.5)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 56
  Identification: 0x0042 (66)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 128
  Protocol: UDP (0x11)
  Header checksum: 0xc16 [correct]
    Source: 130.230.52.139 (130.230.52.139)
    Destination: 130.230.52.5 (130.230.52.5)
  User Datagram Protocol, Src Port: 1034 (1034), Dst Port: 53 (53)
  Domain Name System (query)
0000  00 14 5e 28 2e 6a 00 06 1b d3 25 19 08 00 42 00  ..^(.).. ..%...E.
0010  00 33 00 04 04 00 00 00 00 00 00 00 00 00 00 00  3B.B.....:.....
0020  84 03 04 0a 00 35 00 24 f2 78 3a 51 01 00 00 01  4...$.$.x:Q....
0030  00 00 00 00 00 00 03 77 77 77 03 74 75 74 02 66  .....w ww.tut.f
0040  69 00 00 01 00 01  i.....
```

trace file is dns.pcap, from Roman Dunaytsev’s site <http://www.cs.tut.fi/~dunaytse/>

© David Morgan 2009

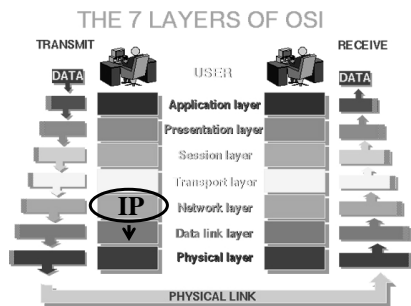
“16 bit one's complement of the one's complement sum of all 16 bit words in the header”

all the 16-bit words in the header (but one!)	cumulative 1's complement sum	
4500	4500	
0038	4538	
0042	457A	
0000	457A	
8011	C58B	
0000*	C58B	
82E6	14871 -> 4872	
348B	7CFD	
82E6	FFE3	1's complement of result
3405	133E8 -> 33E9	CC16

*packet trace shows CC16 here, but that value was unavailable at calculation time (being its outcome), nor therefore available here

© David Morgan 2009

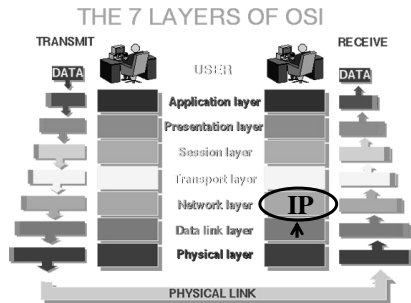
Who is the IP header sender and what does he send?



- inserts calculated result into checksum field (6th word) of header
- sends whole packet with that header

© David Morgan 2009

Who is the receiver and what does he do with received header ?



- adds up all its 10 words
- checks that sum is FFFF
- if not, discards packet

© David Morgan 2009

Why FFFF*?

- addition's terms are header's words
- checksum term, and the other 9 terms' sum, are bitwise inverses (by deliberate design)
- bitwise inverses add up to 1 in each bit position

*FFFF is 1111111111111111

© David Morgan 2009

Performing the check

all the 16-bit words in the header	cumulative 1's complement sum
4500	4500
0038	4538
0042	457A
0000	457A
8011	C58B
CC16	191A1 -> 91A2
82E6	11488 -> 1489
348B	4914
82E6	CBFA
3405	FFFF

© David Morgan 2009

What this mechanism doesn't do

- check anything other than the IP header*
- correct wrong ones

*although, parenthetically, other checksum mechanisms separately might. e.g., the mechanism in udp, tcp, or icmp if one of them is involved.

© David Morgan 2009

Further information

- <http://www.cs.tut.fi/~dunaytse/> website of Roman Dunaytsev at Tampere University of Technology
- Data Communications and Networking, Behrouz Forouzan, McGraw-Hill, Chapter 9 “Error Detection and Correction”
- <http://www.sbprojects.com/projects/tcpip/theory/theory14.htm>

© David Morgan 2009

“16 bit one's complement of the one's complement sum of all 16 bit words in the header”

all the 16-bit words in the header	cumulative 1's complement sum		
4500	4500	homework solution, hidden	
0048	4548		
5CBC	A204		
0000	A204		
8011	12215 -> 2216		
0000	2216		
82E6	A4FC		
3405	D901		
82E6	15BE7 -> 5BE8		1's complement of result
348B	9073		

© David Morgan 2009