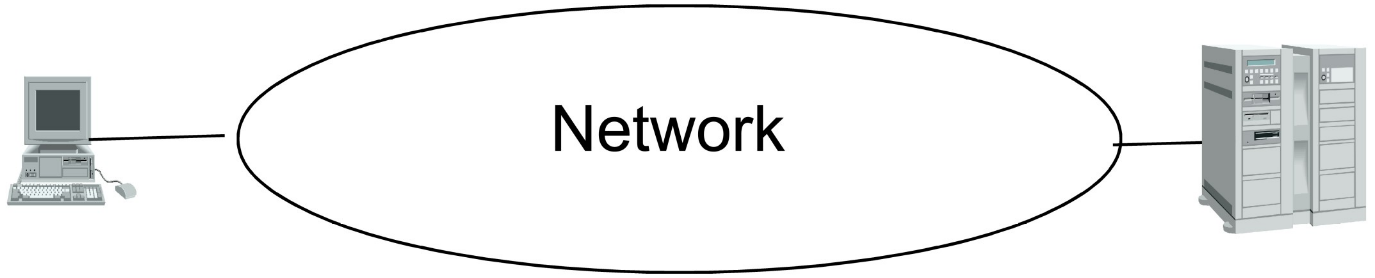
Protocol 3

- How can we provide a reliable service in the transport layer

- Hypotheses
  1. The application sends small SDUs
  2. The network layer provides a perfect service
     1. Packets corruption are possible
     2. No packet is lost
     3. There is no packet reordering
     4. There are no duplications of packets
  3. Data transmission is unidirectional

Transmission errors

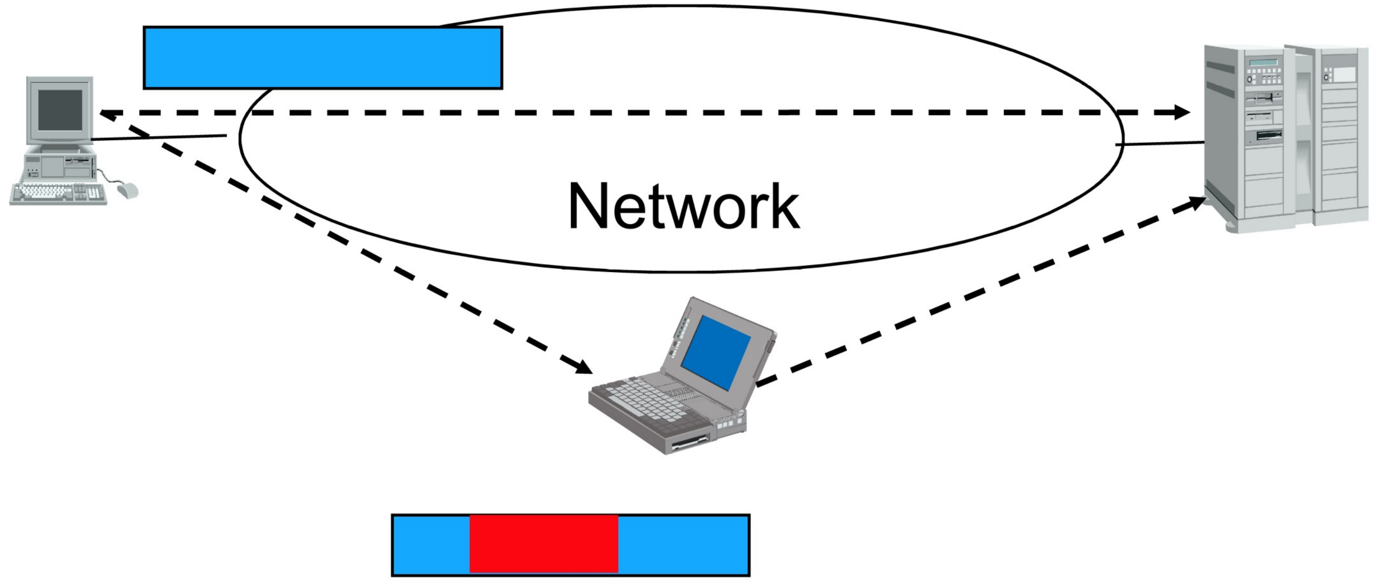
- Which types of packet corruption errors do we need to consider in the transport layer ?



- Physical-layer transmission errors caused by nature
  - Random isolated error
    - one bit is flipped in the segment
  - Random burst error
    - a group of n bits inside the segment is errored
      - most of the bits in the group are flipped

Security issues and packet corruption errors

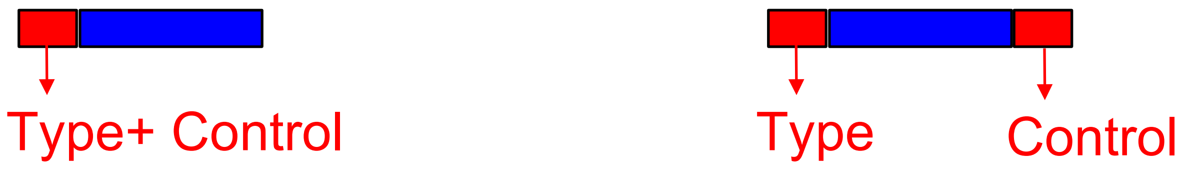
- Information sent over a network may become corrupted for other reasons than transmission errors



- These attacks are dealt by using special security protocols and mechanisms outside the transport layer

How to detect packet corruption ?

- Principle
  - Sender adds some control information inside the segment
    - control information is computed over the entire segment and placed in the segment header or trailer



  - Receiver checks that the received control information is correct by recomputing it

Parity bits

- Simple solution to detect transmission errors
- Used on slow-speed serial lines
  - e.g. modems connected to the telephone network
- Odd Parity
  - For each group of n bits, sender computes the n+1th bit so that the n+1 group contains an odd number of bits set to 1
    - Examples

```
0011010 0          11011001
```

- Even Parity

https://en.wikipedia.o

Internet checksum

- Motivation
  - Internet protocols are implemented in software and we would like to have efficient algorithms to detect transmission errors that are easy to implement
- Solution
  - Internet checksum
    - Sender computes for each segment and over the entire segment (protocol header included) the 1s complement of the « one complement sum » of all the 16 bits words in the segment, with the checksum field to zero.
    - The result is inserted in then the checksum field of the protocole header
    - Receiver recomputes the checksum over each received segment by redoing the « one complement sum » of all 16 bits words, checksum included and check that the sum is correct, i.e that the result is zero in one complement arithmetic. If not, the message is assumed to be corrupt.

Internet checksum

« One complement sum » : Add the 16-bit values up. Each time a carry-out (17th bit) is produced, swing that bit around and add it back into the LSb (one's digit).

This not really a one complement sum as the addition operation used simply ignores register capacity overflow and carry on the computation until there are no more 16 bits words to eat.

```
    -103        1001|1000
    -65         1011|1110
  ________    ___________
            1   0101|0110
             \_________↗

 = -168≠       0101|0111   (87)
```

Internet checksum : example

- Assume a segment composed of 48 bits

```
 0110|0110|0110|0110  0101|0101|0101|0101 0000|1111|0000|1111
+0101|0101|0101|0101
=1011|1011|1011|1011
                                         0110|0100|0110|0111
+0000|1111|0000|1111                    +0101|0111|0101|0100
=1100|1010|1100|1010                    =1011|1011|1011|1011
 0011|0101|0011|0101                    +0000|1111|0000|1111
                                        =1100|1010|1100|1010
                                         0011|0101|0011|0101
```

- Does this detect ALL corrupted messages ?
- Why is there a 1s complement at the end ?

Internet checksum : C implementation

```
{
        /* Compute Internet Checksum for "count" bytes
         *         beginning at location "addr".
         */
    register long sum = 0;

     while( count > 1 )  {
        /*  This is the inner loop */
            sum += * (unsigned short) addr++;
            count -= 2;
    }

        /*  Add left-over byte, if any */
    if( count > 0 )
            sum += * (unsigned char *) addr;

        /*  Fold 32-bit sum to 16 bits */
    while (sum>>16)
        sum = (sum & 0xffff) + (sum >> 16);

    checksum = ~sum;
}
```

Source : RFC 1701 (septembre 1988)

Simple binary division « checksum »

- Motivation & Principle
  - Improve the error detection performances of the Internet checksum by using the division operation instead of the addition + 1s complement.
- Solution
  - Use the division operation instead of the addition + 1st complement
  - Sender and receiver agree on a common divisor.
  - Sender compute the remainder of the message divided by the divisor.
  - Receiver do the same and check if the remainder is the same as the one computed by the sender. If not, the message is assumed to be corrupt.

Example

Assume a segment of 12 bits : 1100 0010 1111
And a divisor of 4 bits : 1001