# CENG393 Computer Networks
# Labwork 5

## IPv4 Packet Header

## 1 Packet Header Structure

Every IPv4 packet has a header that follows the standard definition provided in RFC791 [1] and has the following fields:

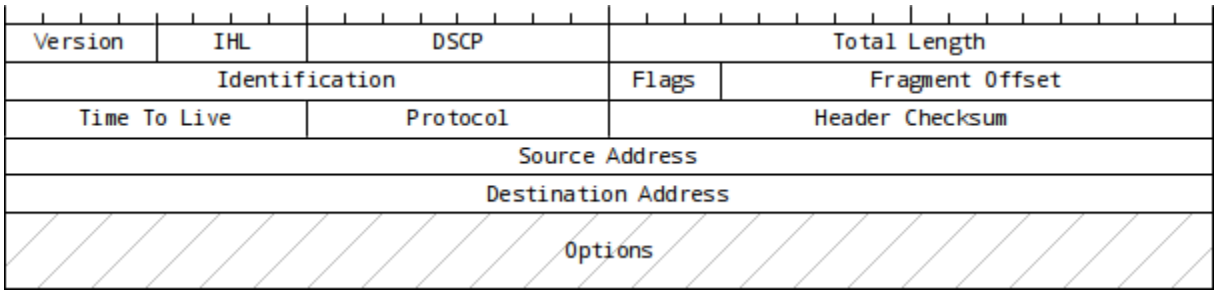


Figure 1: IPv4 Packet Header

- **Version:** Version of protocol. For IPv4, its value is 4.
- **IHL:** Header length. The minimum value is 5 (# of 32-bit words).
- **DSCP:** Differentiated Services Code Point. Used for new services that, for example, require real-time data streaming.
- **Total Length:** Length of the whole IP packet (data included).
- **Identification:** Numerical identification of a packet.
- **Flags:** Reserved (always 0), Don't Fragment (DF), More Fragments (MF).
- **Fragment Offset:** Offset of a fragment relative to the beginning of the complete packet.
- **Time To Live:** Specifies how many routers this packet can traverse through until it is expired.
- **Protocol:** Used to define the protocol used in the data.
- **Header Checksum:** Error checking field for the header. Must be calculated and entered after all other fields in the header.
- **Source Address:** IP address of source device.
- **Destination Address:** IP address of destination device.
- **Options:** Used to modify routing behaviors, experimental features, etc.

# 2 IPv4 Header Checksum

The checksum field in the header is used for verifying if the header has been transmitted without problems or not. It is calculated by the source device before transmission and it is used for verification by destination device before processing the packet data.

## 2.1 Calculation

In order to calculate the checksum, all information except the checksum must already be available in the header. For example lets assume the following sequence is a 20 bytes long IPv4 header (marked bytes belong to checksum field):

```
45 00 00 30 0f 41 40 00 80 06 00 00 91 fe a0 ed 41 d0 e4 df
```

1. Calculate sum of every 16-bit words in header.

   ```
   = 4500 + 0030 + 0f41 + 4000 + 8006 + 0000 + 91fe + a0ed + 41d0 + e4df
   = 00036e11
   ```

2. If the result is longer than 16 bits, divide it into 16-bit words and add them together. Repeat this step as many times as necessary (especially if options are present).

   ```
   = 0003 + 6e11
   = 6e14
   ```

3. Calculate 1's complement of the value you have found in step 2.

   ```
   = ~6e14
   = 91eb
   ```

After this calculation, the result is written in its place in the header and the packet is sent on its way:

```
45 00 00 30 0f 41 40 00 80 06 91 eb 91 fe a0 ed 41 d0 e4 df
```

## 2.2 Verification

When the next device in the network acquires this packet, it first checks the header if it has been received correctly or not. In order to do so, this device follows the same calculation algorithm, but this time the data used by the device already contains the checksum:

1. Calculate sum of every 16-bit words in header.

   ```
   = 4500 + 0030 + 0f41 + 4000 + 8006 + 91eb + 91fe + a0ed + 41d0 + e4df
   = 0003fffc
   ```

2. If the result is longer than 16 bits, divide it into 16-bit words and add them together. Repeat this step as many times as necessary.

   ```
   = 0003 + fffc
   = ffff
   ```

3. Calculate 1's complement.

   ```
   = ~ffff
   = 0000
   ```

If the result is zero, that means the data in the header can be assumed as correctly received. For any other value, it is assumed that either the checksum or the other fields in the header have been corrupted during transmission and such packets are discarded.

Do note that if the packet has not arrived to its destination network yet, the receiving router will alter the header (e.g. TTL will be decreased); therefore each receiving router must calculate a new checksum value until the packet has reached its destination network.

# 3 Exercises

1. Study the following code:

```
#include <stdio.h>
int main() {
  int i, byte_grp, header[20] = { 0x45, 0x00, 0x00, 0x30, 0x0f, 0x41, 0x40,
    0x00, 0x80, 0x06, 0x00, 0x00, 0x91, 0xfe, 0xa0, 0xed, 0x41, 0xd0, 0xe4,
    0xdf };

  for (i = 0; i < 10; i++) {
    byte_grp = (header[2 * i] << 8) + header[2 * i + 1];
    printf("%04x ", byte_grp);
  }
  return 0;
}
```

   Complete this code so that it can calculate and print the checksum of the given header. In order to do bitwise calculations, you may use the operators demonstrated in Appendix A below.

2. In Linux command line interface, **ping** command can be used to send ICMP_ECHO_REQUEST messages between hosts and **tracepath** (or **traceroute** if installed) command can be used to print the route to a specified destination. Choose any host you want and use these commands to observe the output. Then using man documentation, find out which options are used for changing TTL value and use these commands again with lower TTL values. What difference do you see now?

# Appendices

## A Bitwise Operators in C

```
&  : Bitwise AND                |  : Bitwise OR
^  : Bitwise XOR                ~  : Bitwise NOT
>> : Bitwise Shift Right        << : Bitwise Shift Left
```

## B Sample Program for Bitwise Operators

```
#include <stdio.h>

int main() {
  //  8-bit variables: char      or unsigned char
  // 16-bit variables: short int or unsigned short int
  // 32-bit variables: int       or unsigned int
  // 64-bit variables: long int  or unsigned long int

  // decimal representation    : 116
  // hexadecimal representation: 0x74
  // octal representation      : 0164
  // binary representation     : 0b1110100

  unsigned int a = 0x6D;
  unsigned int b = 60;

  printf("a & b     = 0x%02X (%u)\n", a & b, a & b);
  printf("a | b     = 0x%02X (%u)\n", a | b, a | b);
  printf("a ^ b     = 0x%02X (%u)\n", a ^ b, a ^ b);
  printf("~a        = 0x%02X (%u)\n", ~a, ~a);
  printf("~a & 0xFF = 0x%02X (%u)\n", ~a & 0xFF, ~a & 0xFF);
  printf("a << 1    = 0x%02X (%u)\n", a << 1, a << 1);
  printf("b >> 2    = 0x%02X (%u)\n", b >> 2, b >> 2);

  return 0;
}
```

## References

[1] IETF Tools. Rfc791 - internet protocol. https://tools.ietf.org/html/rfc791, 1981.