Chapter Five

IP Protocol – IP packet
Network Layer

Data Communications and Computer Networks

Introduction

The Internet Protocol IP:

The Internet Protocol (IP) is a fundamental Network Layer protocol (Layer 3) that routes data packets across networks.

It is connectionless, meaning each packet is sent independently without first setting up a dedicated path (a phone call requires a connection before talking, IP just sends data like mailing letters).

IP is also unreliable, it doesn't guarantee packets will arrive in order, or avoid duplication (just like letters might get lost or arrive late in the mail). This makes IP fast and simple.

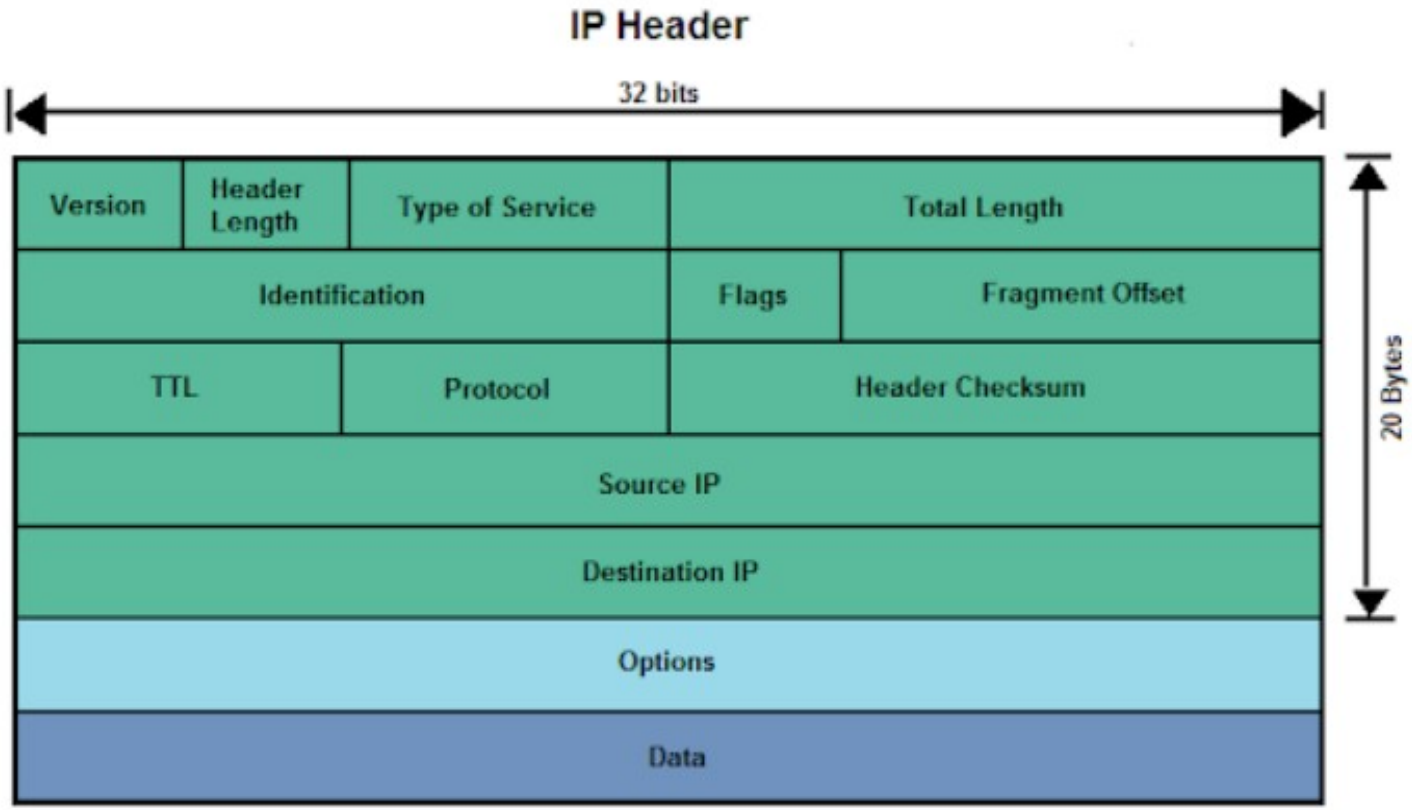
The Internet Protocol IP

For reliable delivery, protocols like TCP (runs on top of IP) fixes IP reliability issues by checking for errors, resending lost data, and putting packets in order.

If an IP packet encounters error the IP Packet is discarded and an *ICMP error message* is sent back to the source.

The key idea is that IP provides best-effort delivery, it does its best to send data, but doesn't guarantee packets arrive successfully.

IP Header


IP Header
32 bits
Version
Header Length
Type of Service
Total Length
Identification
Flags
Fragment Offset
TTL
Protocol
Header Checksum
Source IP
Destination IP
Options
Data
20 Bytes

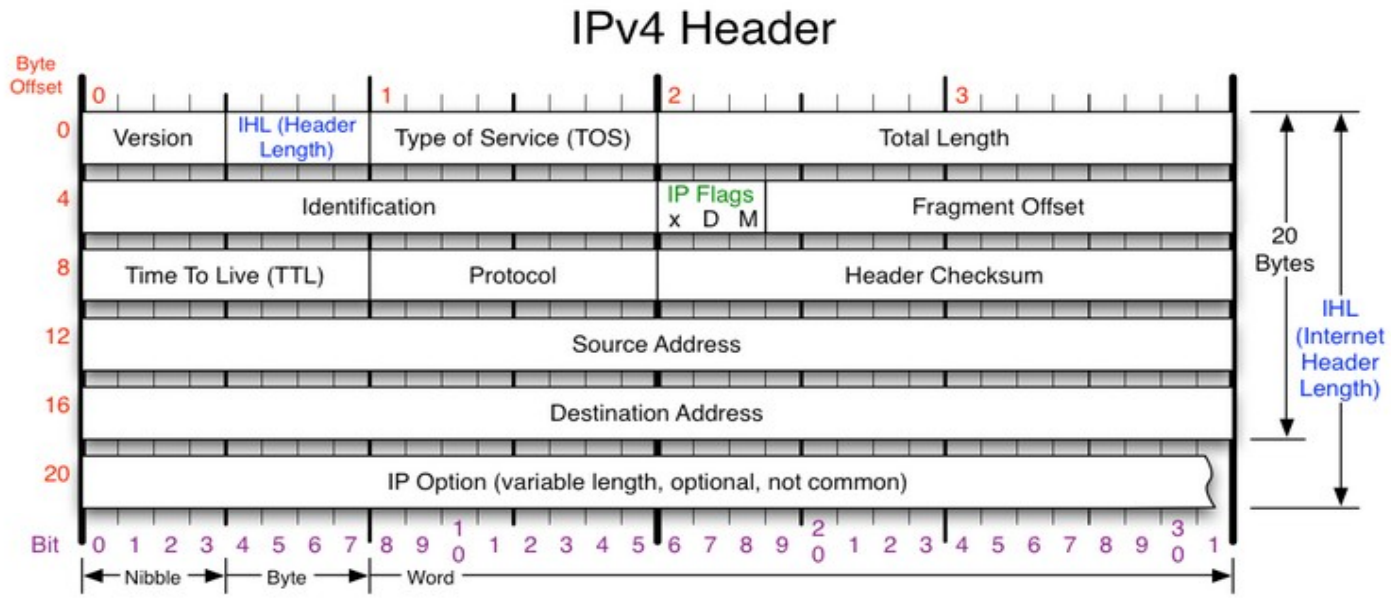
IP V4 Header

IP header is prefixed to every IP packet, 20 to 60 bytes in size, it contains all the necessary information for routers to deliver the packet to its destination.

Key fields:

- The source and destination IP addresses (where the packet is coming from and going to),
- The Time-To-Live or TTL (limits how long the packet can circulate to prevent endless looping),
- The Protocol field (indicates whether the payload contains TCP, UDP, or other upper-layer data),
- and a Header Checksum (used to detect errors in the header).

IP Header



Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

Header Checksum

Checksum of entire IP header

IP Flags

x D M

x 0x80 reserved (evil bit)
D 0x40 Do Not Fragment
M 0x20 More Fragments follow

RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

IP Header

The first 32 bits (4 bytes) of the IPv4 header:

Version: (4 bits) specifies the IP protocol version, 4 indicates IPv4 and 6 indicates IPv6.

Internet Header Length (IHL - 4 bits) defines the header size in 32-bit units, minimum value of **5** (20 bytes) for standard headers and up to **15** (60 bytes) if options are included.

Type of Service ToS: (8 bits) is used for Quality of Service.

Total Length: (16 bits) tells the entire packet size, including both header and data, in bytes. The minimum size is 20 bytes (if you have no data).

The maximum size is 65535 bytes.

In IP header context, 1 word = 32 bits

IP Header

The first 32 bits (4 bytes) of the IPv4 header

Ex:

Version: 0100 | IHL: 0101 | ToS: 00000000 | Total Length: 0000000000101100

Version (4 bits): 4 (IPv4)

IHL (4 bits): 5 (20-byte header)

ToS (8 bits): 0 (no priority)

Total Length (16 bits): ex: 44 bytes (20 header + 24 data)

IP Header

Good to know

ToS is obsolete today

Replaced by DSCP Differentiated Services (DiffServ)

In modern IPv4 headers, the 8-bit ToS field is split into just two functional parts:

DSCP (6 bits, bits 0-5): Defines QoS priority

ECN (2 bits, bits 6-7): Signals congestion feedback

DSCP Differentiated service code point and ECN Explicit congestion notification

IP Header

Configure routers/switches to mark traffic with these values:

ECN values (2 bits: 4 possible values)
00: Non-ECN-Capable.
01 or 10: ECN-Capable (sender supports ECN).
11: Congestion Experienced (Ex: router has experienced congestion).

DSCP values (6 bits):
000000 (0): Best Effort (default, no priority).

101110 (46): EF (Expedited Forwarding)
For VoIP/video calls (low latency, high priority).

AF Classes (Assured Forwarding):
AF41 (100010/34): High-priority video (Ex: streaming).
AF31 (011010/26): Medium-priority (Ex: business apps).

Ex: in ToS field we can put: 10111000. first six bits for DSCP (VoIP), last two bits for ECN (usually 00)

IP Header

Identification field:

The 16-bit Identification field uniquely marks a packet so its fragments can be reassembled correctly by the receiver.

When a packet is split, all fragments keep the same Identification value (Ex: 0xA3F1).

The sender can set this field's value to a random number or as an incrementing counter, but it must remain unique for the packet's lifetime (usually seconds) to avoid mixing fragments from different packets.

IP Header

Flags(3 bits):
first bit is **unused**, always **0**, **second** bit if fragment is allowed or not: 0 if allowed and 1 if not allowed, **third** bit indicates incoming or last fragment: 0 for last and 1 for more fragments coming.

000 = this is the last fragment (Fragments allowed). DF =0 ,MF= 0
001 = more fragments coming (Fragments allowed). DF=0 ,MF = 1

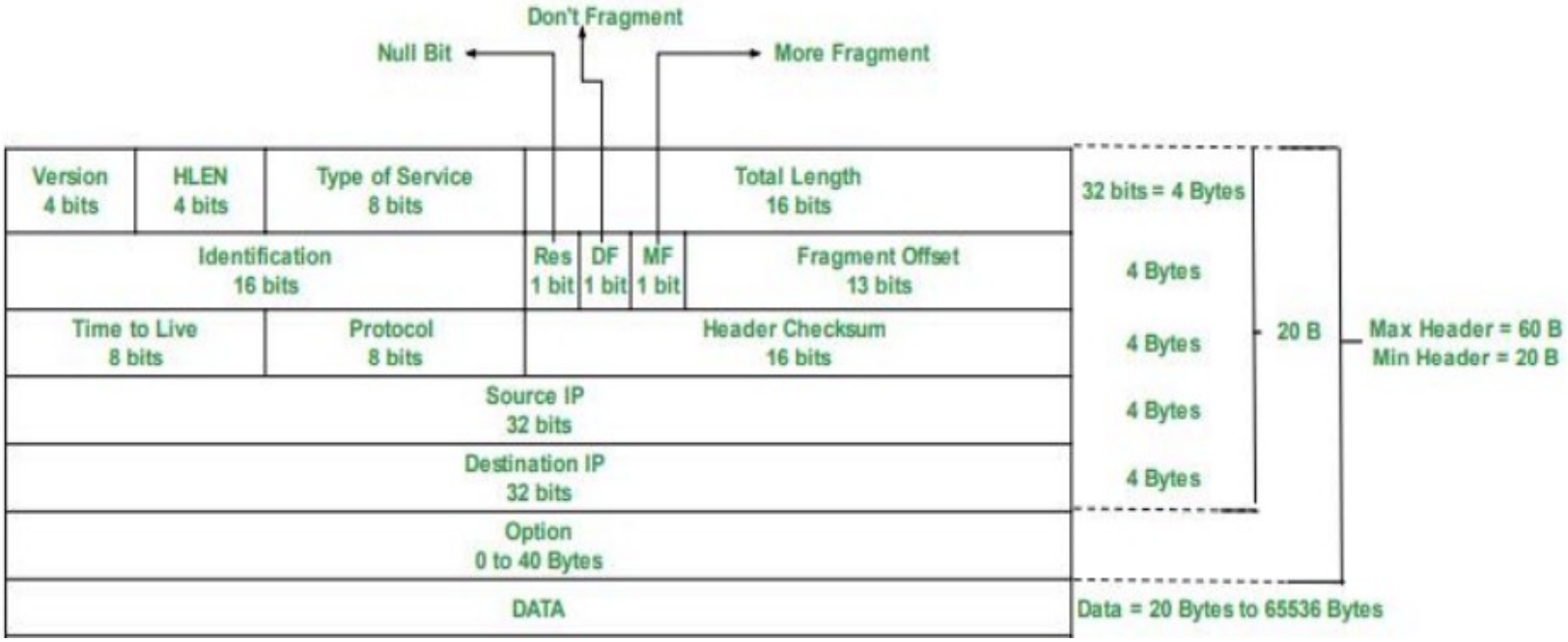
010 = DF "Don't fragment" (drop if too big). DF= 1 , MF= 0

Example:
When sending fragments:
For all fragments (except the last one) we use 001. The last fragment uses 000 to signal: "No more fragments, reassemble now"

Fragment 1: 001 (MF=1)
Fragment 2: 001 (MF=1)
Fragment 3: 000 (MF=0) "Last part, reassemble now"

IP Header

Flags(3 bits):



IP Header

Fragment Offset (13 bits):

Specifies the position of each fragment in the original unfragmented packet, ensures the receiver can reassemble fragments in the correct order.

Measured in "8-byte" **blocks** (not bytes)

First fragment always has offset 0.

Maximum value: 8191 fragments (since $2^{13}-1=8191$) fragment offset is 13 bits long.

Covering packets up to 65,528 bytes (8191×8).

IP Header

Ex:
We are sending a 4,000-Byte Packet (MTU = 1,500 Bytes (Maximum Transmission Unit) for Ethernet is 1500 bytes. (you have to deduct the IP header 20 bytes)

Important: Each fragment needs a 20-byte IP header.

Max payload per fragment = MTU (1,500) - Header (20) = 1,480 bytes.

Offset = (Starting byte of payload) ÷ 8

Let's calculate the Fragments:
Original data: 4,000 bytes.

Fragment 1:
Payload: 1,480 bytes (bytes **0** - 1,479 of original data).
Total size (MTU): 1,480 (payload) + 20 (IP header) = 1,500 bytes.
Offset: **0** / 8 = 0. (Starting byte of payload 0 ÷ 8)

IP Header

Fragment 2:
Payload: 1,480 bytes (bytes **1,480** - 2,959).
Total size (MTU): 1,480 + 20 = 1,500 bytes.
Offset: **1,480** / 8 = 185. (Starting byte of payload 1480 ÷ 8)

Fragment 3:
Remaining data: 4,000 - (1,480 × 2) = 1,040 bytes (bytes **2,960** - 3,999).
Total size (MTU): 1,040 + 20 = 1,060 bytes.
Offset: **2,960** / 8 = 370. (Starting byte of payload 2960 ÷ 8)

Fragment	Payload Size	Offset	Flags	
1	1,480	0	001	(more fragments coming)
2	1,480	185	001	(more fragments coming)
3	1,040	370	000	(last fragment)

370 (offset) x 8 = 2960, the starting point of the last fragment in the original packet and ends at 3999. Total of 4000 bytes, divided as 3 fragments of 1480,1480,1040 bytes each.

IP Header

FYI:

IP headers (20 Bytes) are added on top of the payload during transmission (overhead).

Original: 4,000B data
Fragmented: 4,000B data + 60B headers = 4,060B total.

TTL Time To Live

TTL Time To Live

•Prevents packets from looping forever.

•The sender set the value based on the OS used, ex: windows 128, Linux 64 and Cisco router 255.

•Each ***router*** decrements TTL by 1 before forwarding, if TTL reaches 0, the packet is dropped, and an ICMP “Time Exceeded” message is sent back.

Used for “traceroute”.

TTL Time To Live

How “traceroute” reaches every router: (intentionally sending probes with low TTL values)

Initially: traceroute sends a packet (ex: UDP, ICMP, TCP) with TTL=1

First router decrements TTL=1 to 0, drops it, and sends back an ICMP Time Exceeded error (with its own IP). Now we know the first hop

Source increments to TTL=2: Next packet reaches the first router (TTL=2 decrement by 1, the TTL is not zero yet), so the packet is forwarded to the second router.

Second router decrements TTL=1 to 0, drops it, and replies with its own IP. Now we know the second hop

The source repeats this, Increasing TTL +1 each time and continues until a packet finally reaches the destination and replies with its own IP.

This is how traceroute maps the entire path from source to destination

TTL Time To Live

TTL Time To Live

Ex:
use command prompt cmd: traceroute google.com

Source → Router1 → Router2 → ... → Router N → Destination
(TTL=1) (TTL=2) (TTL=N+1)

IP Header

Protocol (8 bits)

Tells the destination what data is inside the IP packet.
Its like a label. The most common values are:

6 = TCP (reliability, connection-oriented – used for web (help rebuild streams), email).
17 = UDP (fast, connectionless – used for video calls like zoom, DNS).
1 = ICMP (ping, handles traceroute errors).
2 = IGMP (multicast traffic).

Header Checksum

The Header Checksum (16 bits) detects errors to ensure the IP header's integrity.

Before sending, the source device calculates a checksum value.

The receiver repeats the same calculation; if the result doesn't match the checksum value, the header is corrupted, and the packet is discarded.

Hint: This checksum only protects the header (not the data) and operates at Layer 3, meaning errors are caught but not fixed (retransmission relies on higher layers like TCP).

This checksum prevents corrupted headers from causing routing errors or security issues.

Header Checksum

Checksum error example:

Step 1: combine fields into 16-bit groups
01000101 00000000 (version + IHL + ToS)
00000000 00111100 (total length)
10100011 11110001 (identification)
01000000 00000000 (flags + fragment offset)
01000000 00000110 (TTL + protocol)
Step 2: Binary sum
0100010100000000 (version/IHL/ToS)
+ 0000000000111100 (total length)
= 0100010100111100
+ 1010001111110001 (identification)
= 1110100100101101
+ 0100000000000000 (flags/offset)
= 0010100100101101
+ 0100000000000110 (TTL/protocol)
= 0110100100110000 (total sum)

Header Checksum

Checksum error example:

Step 3: 1's complement, invert bits

0110100100110000 → 1001011011001111 (sender checksum)

Step 4: verification (at receiver)

0100010100000000 (version/IHL/ToS)

+ 0000000000111100 (total length)

+ 1010001111110001 (identification)

+ 0100000000000000 (flags/offset)

+ 0100000000000110 (TTL/protocol)

+ 1001011011001111 (sender checksum)

= 1111111111111111 (all 1's = header valid)

If any bit is 0 this means that the header is corrupted.

Hint: simply add the bits at step 3, no need to sum all fields again, if you get all 1's means header is valid, otherwise it's a corrupt header (header with errors).

Header Checksum

Checksum error example:
FYI:

Binary addition:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 (write 0 and carry 1 to next column)
For simplicity try this 4-bit summation: (6+3=9)
0110
+ 0011
= 1001

Wireshark

Wireshark example to check IP traffic:

Open wireshark, choose interface (wifi or ethernet)
Click start
In the filter bar type: ip (this shows ip traffic)
Click any ip packet (http, dns etc)
in the middle panel, expand internet protocol version 4 (IPv4)
Here we can see all fields of the ip header

For checksum verification:
Edit then preferences then protocols then IPv4 then click validate checksum.

Summary

- IP protocol
- Structure and Fields – Version, IHL, ToS, Length, ID, Flags, TTL, Protocol, Checksum, etc.
- Fragmentation: Identification, Flags (DF/MF), Offset for splitting/reassembling packets.
- Checksum Calculation – 16-bit binary sum + carry wrap + inversion for error detection.
- TTL (Time to Live) – Limits hops, prevents loops, used in traceroute.
- Protocol Field – Identifies TCP (6), UDP (17), ICMP (1), etc.
- Wireshark example