IPV4 Header →

<table>
<tr><td>Version(4)</td><td>Header Length(4)</td><td>Type of service(8)</td><td colspan="4">Total Length(16)</td><td>4B</td><td rowspan="5">20B</td></tr>
<tr><td colspan="3">Identification (16)</td><td>0</td><td>D F</td><td>M F</td><td>Fragment Offset (13)</td><td>4B</td></tr>
<tr><td colspan="2">TTL(8)</td><td>Protocol(8)</td><td colspan="4">Header Checksum(16)</td><td>4B</td></tr>
<tr><td colspan="7">Source IP(32)</td><td>4B</td></tr>
<tr><td colspan="7">Destination IP(32)</td><td>4B</td></tr>
<tr><td colspan="7">Options (0 to 40 Bytes)</td><td></td><td></td></tr>
<tr><td colspan="7">Data.</td><td></td><td></td></tr>
</table>

Min. Header = 20B , Max. Header = 60B

$$\boxed{\text{Header Length} = \frac{\text{Actual Header Length}}{4}}$$

Header Length - (20B-60B) → HL field - (5B-15B)

If size is 30B (not divisible by 4), so we put padding 2B (in options) to make it multiple of 4.

Version → V4 or V6   V5 ✗

V4 ↓ Diff. f^ns   V6 ↳ Diff. f^ns

Identificat^n No. → To number, every datagram out of network layer

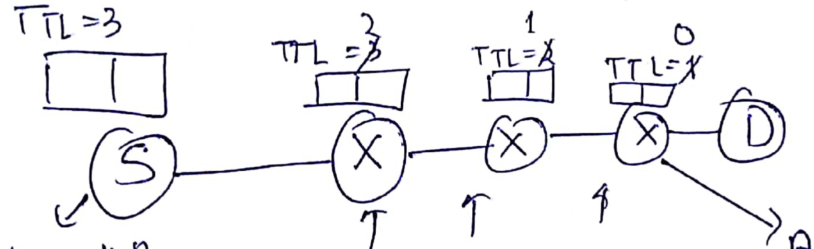
DF → Do not fragment

Entire Datagram has to be sent, no pieces

MF → More Fragments

Fragment Offset → No. of data bytes ahead of this particular fragment in this particular datagram.

TTL (Time to Live) → Restrict the number of hops.
Done to prevent infinite looping.



Sender, says this packet can reach the destⁿ in max 3 hops, if it is not at destⁿ after 3 hops, anyone in the middle has the right to discard it

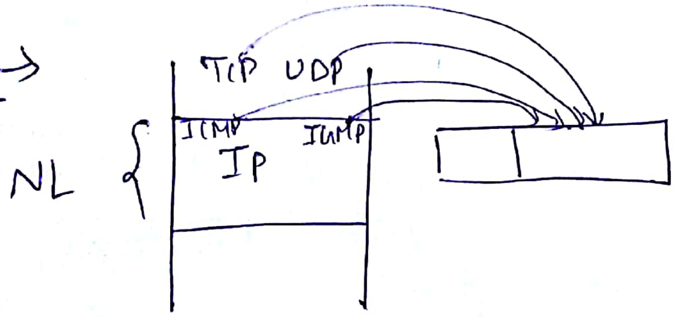
Every router in b/w reduces TTL by '1', showing 1 hop more taken.

Already taken 3 hops, packet will be discarded.

$\boxed{\text{Dest}^n \text{ will accept if } TTL \geq 0}$

Any device, having network layer will reduce TTL by '1'.

Protocol → (8)



Protocol will indicate what is the protocol of the datagram, either it is TCP, UDP, ICMP or IGMP.

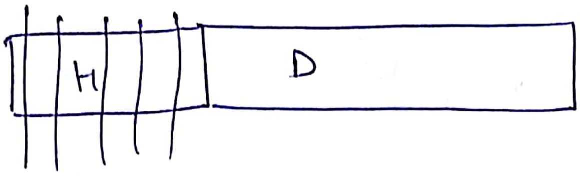
Every protocol will have a number associated with it.

At router, if there is congest^n (say buffer is full), then according to priority of the packets in the buffer & the incoming packet, protocols of the router will decide whether to discard the incoming packet or to discard ~~the~~ a packet in buffer to make space for incoming packet

<u>Priority</u> →

ICMP < IGMP < UDP < TCP.

Header Checksum (16) → Checksum only for Header

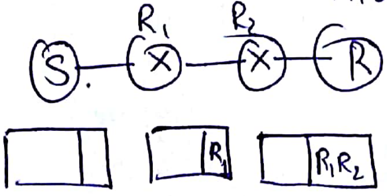


Into 16 parts (for calculation)

Initially, for checksum calculat^n, the field itself is zero.

We calculate it only for header bcoz, at router TTL changes, FO, MF, TL, HC$_n$ change so at every router Header checksum is also calculated, so to reduce calculat^n at router we do not include data in it.

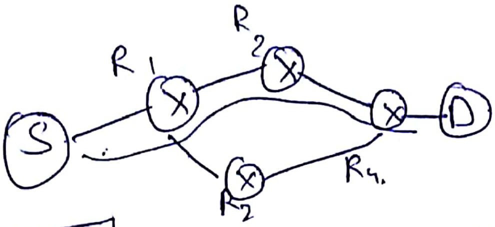
Options → 1) Record route → R_1 & R_2 will be written on the packet, so the receiver knows the path, that the packet took.



Max. No. of IP add. = 9 (recorded)

(2) Source Routing →

Specifying the route, that the packet takes by the sender himself.



D $R_4 R_2 R_1$

Strict Source Routing

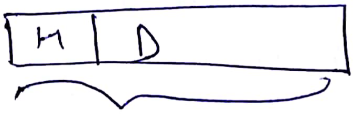
(Specifying Every Hop)

Specify, ~~should~~ R_1 should be taken & then you don't care i.e, Not Every Hop is specified (Loose Source Routing)

Used to test if a path is working or not.

(3) Padding → To make size of packet of packet divisible by '4'.

Total Length → 16 bits = $2^{16}-1$ = 65,535 (Max. Size of IP Datagram)



AL

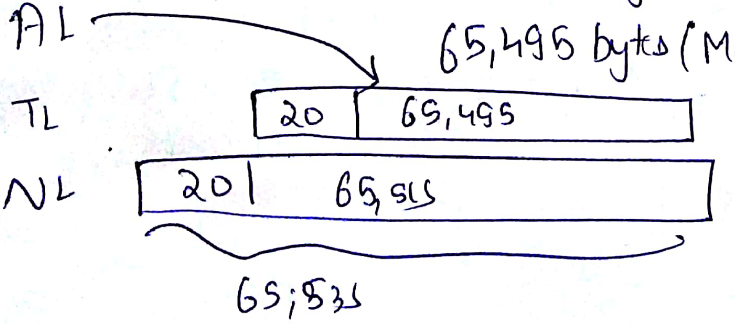
(Segment) TL ← (Application Layer can give any amount of data to TL

Max. segment size = 65,515 (− 20 (TL Header))

(Datagram) NL

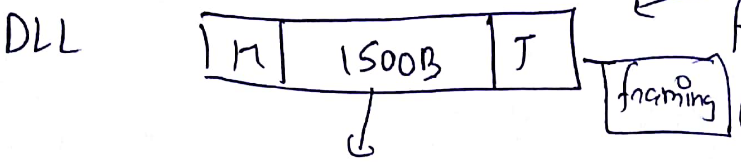
So, Max. Data in a TCP segment

65,495 bytes (Max) — Segmentation



NL, can produce a datagram of max. size 65,535 bytes

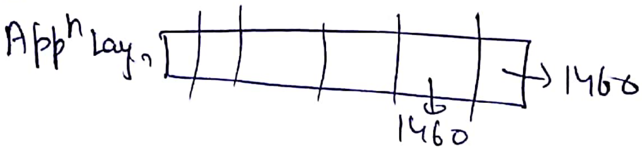
But, DLL can have max payload of 1500 Bytes

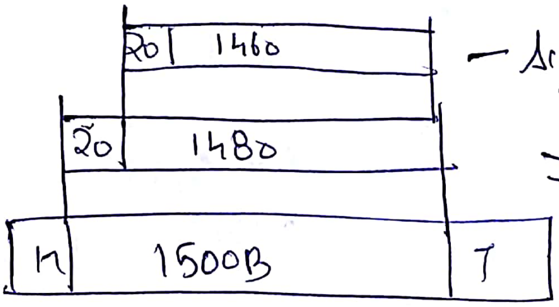


MTU
(Maximum Transmittable Unit)

fragmentat^{n} (Done at NL, dividing Datagram to make it fit into Frame)

Transport Layer, will check which is bottleneck NL or DLL & then it will create segments according to that only.

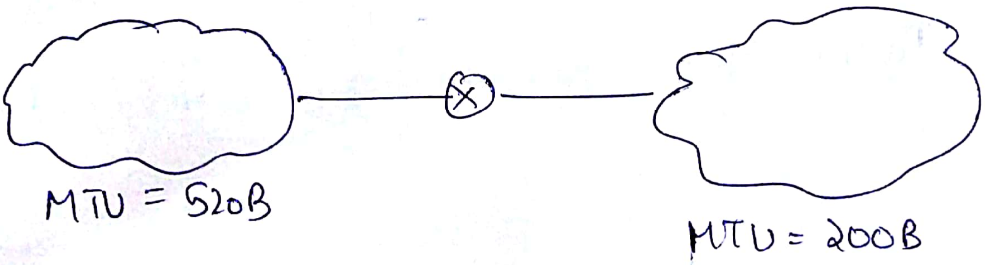


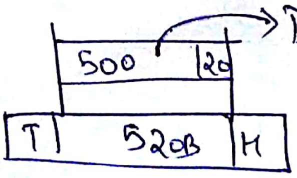


— Segmentat^{n}

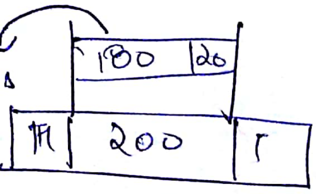
⇒ So no fragmentat^{n} has to be done

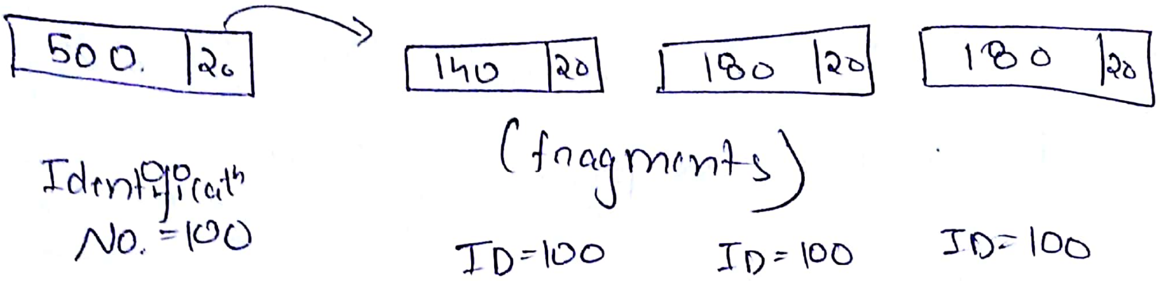
At host, segmentat^{n} is done





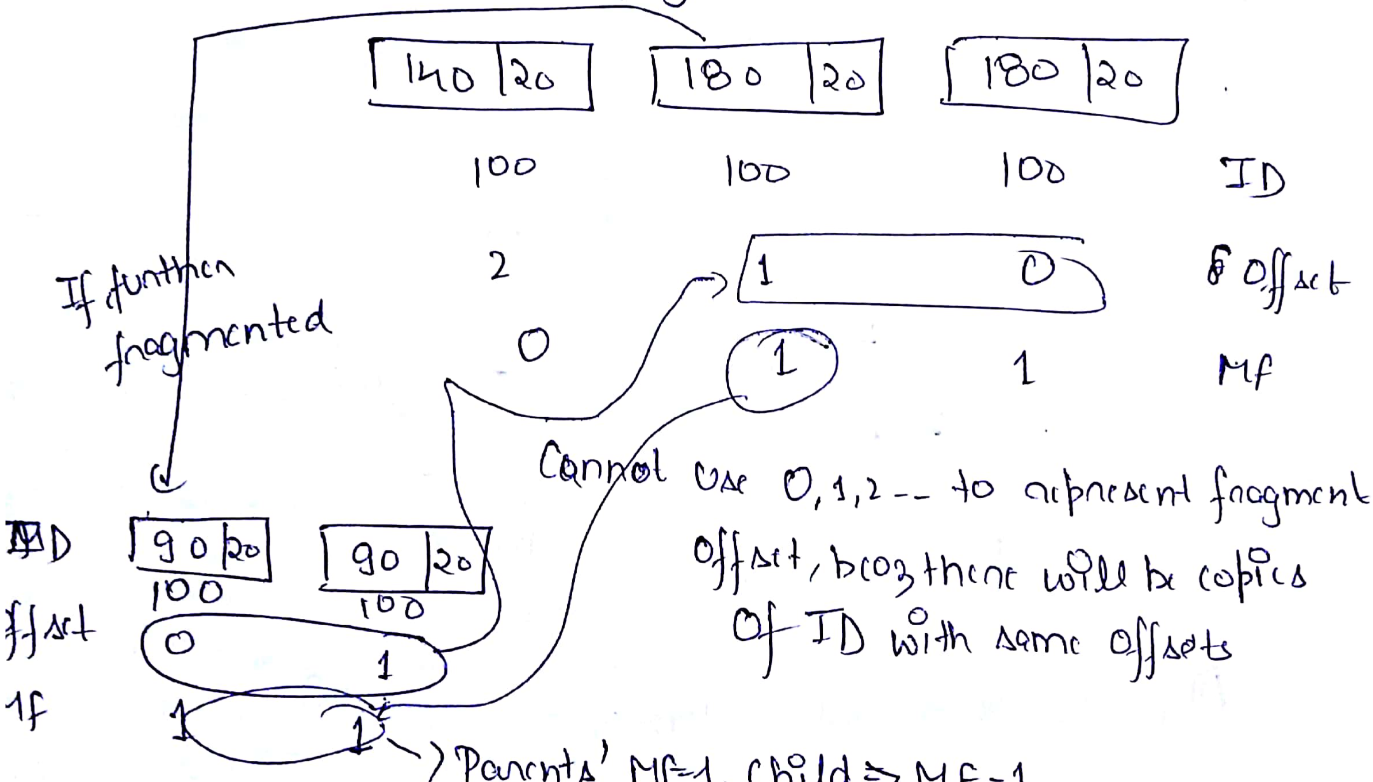
→ Dividing this data to make it in 180 bytes for, the other network, it is k/a fragmentat^{n}





fragment Offset → Used to identify the order of fragments

MF → Whether more fragments are following this fragment

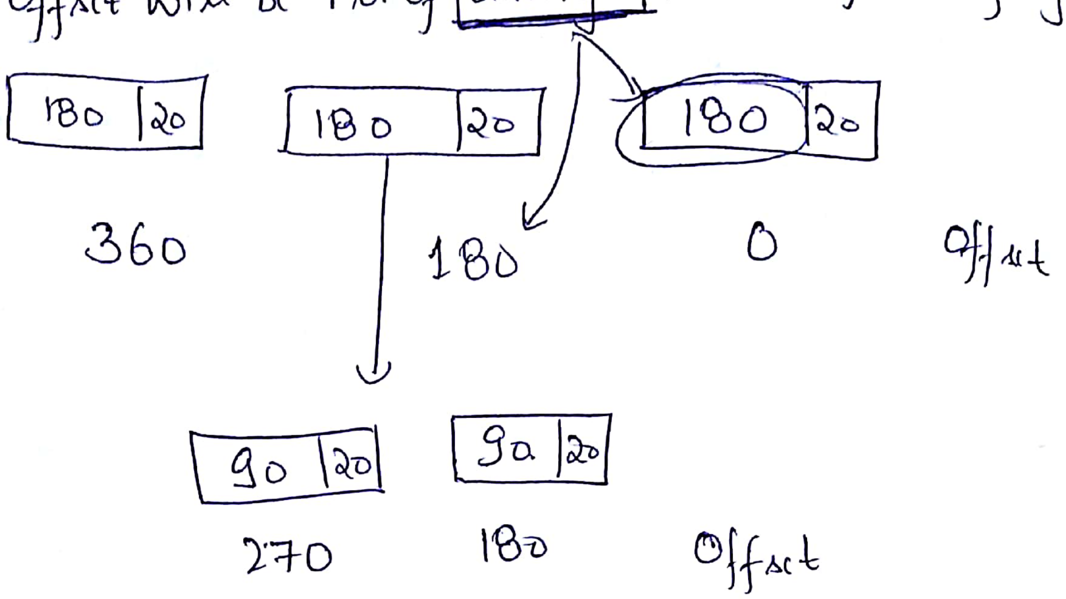


Cannot use 0,1,2 -- to represent fragment offset, bcoz there will be copies of ID with same offsets

> Parents' MF=1, Child => MF=1 (all)

Parents' MF=0, last child => MF=0 rest MF=1

So, Offset will be no. of data bytes, ahead of this fragment.



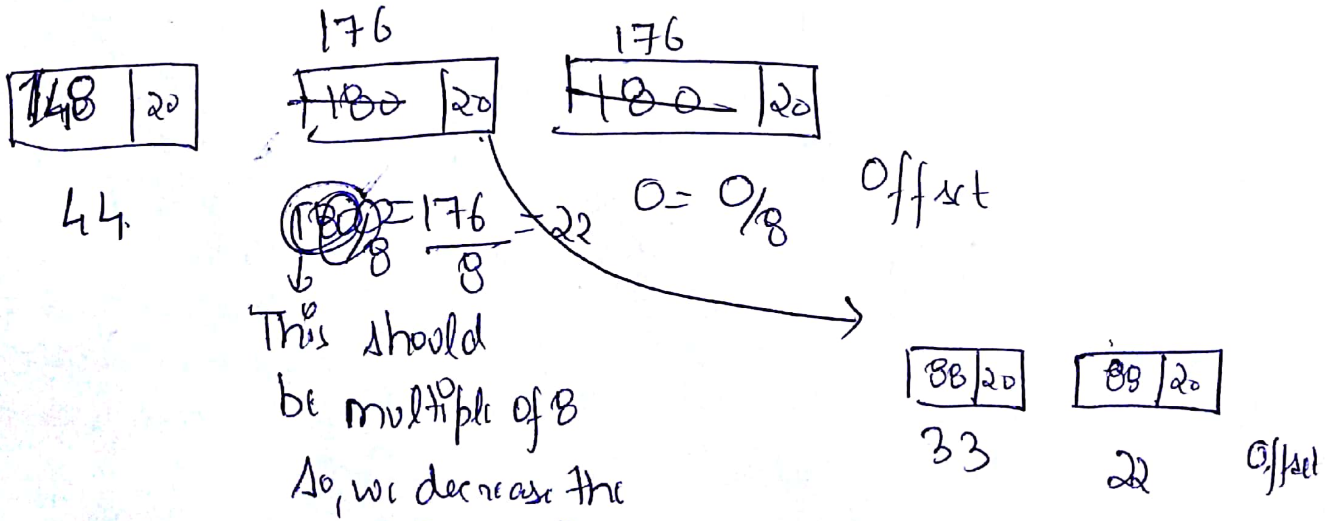
So, receiver gets 4 fragments →

4 (0, 180, 270, 360)

for, reassembly, receiver counts the data bytes in a segment, adds it to the offset & finds the offset of next fragment & combines them.

In worst case, offset ≃ 2^{16}, but offset field = 13 bits

So, we will scale this field by a factor of 8.



This should be multiple of 8
So, we decrease the no. & making it multiple of 8

1 (500B + 20B) datagram → 4 fragments

So, 3 headers are overheads

So, 60B is overhead

$$\eta_{NL} = \frac{UB}{TB} = \frac{500}{500+4\times20} = \frac{50\cancel{0}}{58\cancel{0}}$$

$$\boxed{\text{Throughput} = \eta\ BW}$$

UDP →

i) Applicat^n needs 1 req/1 rep.

1) DNS

2) BOOTP

DHCP

3) NTP

4) NNP

5) quote of day

2) Broadcasting/Multicasting

3) Fastness > Reliability

(Constant Data Rate)

⇒ Multimedia

⇒ Online games.

UDP Header →

Source Port (16)	Dest^n Port (16)
Length (16)	Checksum (16)

↓

UDP Header + Data

User Datagram Protocol.

No Ack in UDP

No flow control in UDP

No flags (No Connect^n Establishment, No connect^n Terminat^n)