**Chapter 07: Internet Protocol Version 4**

Table of Contents

[7.2 Datagrams 3](#_Toc86833794)

[Header 3](#_Toc86833795)

[Service Type 6](#_Toc86833796)

[7.3 Fragmentation 9](#_Toc86833797)

[7.4 Options 12](#_Toc86833798)

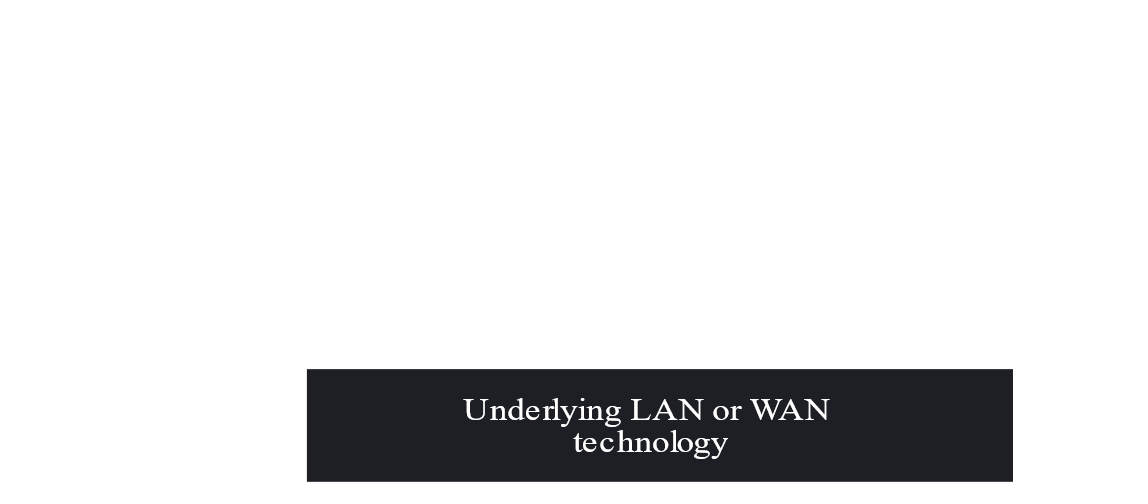
[Record Route 13](#_Toc86833799)

[Source Route 14](#_Toc86833800)

[Timestamp 16](#_Toc86833801)

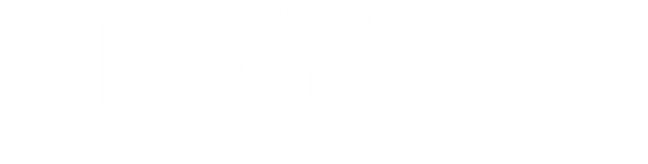
[7.8 IP Package 18](#_Toc86833802)

The **Internet Protocol** (IP) is the transmission mechanism used by the TCP/IP protocols in the **network layer**.



## 7.2 Datagrams

Packets in the network layer are called **datagrams** or **IP packets**. It is a **variable-length** packet consisting of a **header** and **data**.

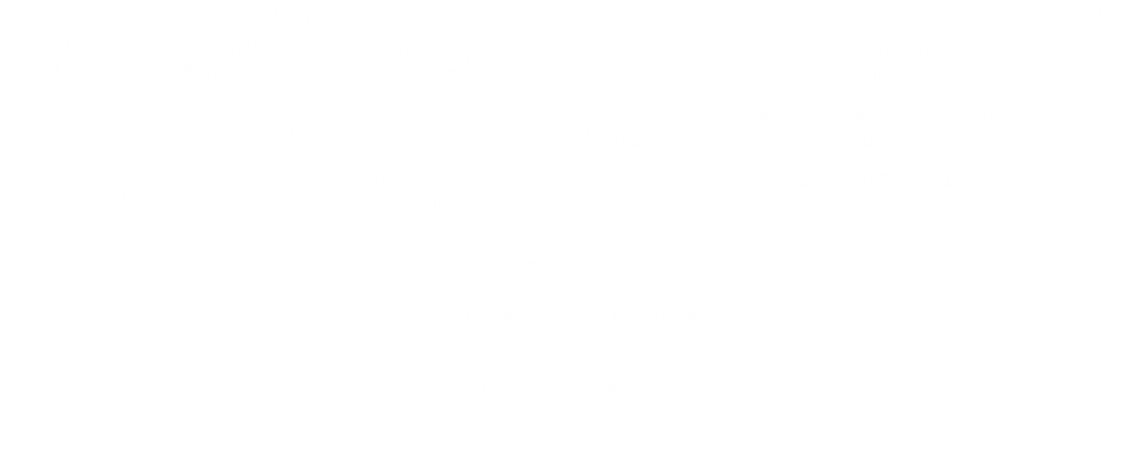


The **header** is **20 to 60 bytes** long and contains information essential to **routing** and **delivery**. The complete datagram is **20 to 65,535 bytes** long.

Notice that the maximum possible size is significantly **larger** than the maximum allowable size of a frame in the data link layer. Datagrams that are too large are broken down into **fragments** and sent in parts.

### Header

We will now look into the contents of the **header**. In TCP/IP, it is customary to show the header as **4-byte sections**. Because of this, the header shown below is divided into rows of 4-bytes each.



* **VER** – The first 4 bits are used to identify the **version**, either IPv4 or IPv6. Since there are 4 bits, there can be versions, but only two are currently used.
* **HLEN** – The next 4 bits are used to provide the **header length**. In reality, the values can be from to . This value is multiplied by to get the actual header length in bytes.

For example, if we get a packet starting with , we can use the first four bits to identify that this is an IPv4 packet and the second four bits to find the length. However, the header cannot be of bytes length, which tells us this packet must be corrupted. Similarly, if we get a packet starting with , the header length is bytes, which is valid, but the VER field seems to be , which does not exist. Again, this packet is corrupted.

* **Service Type** – This refers to how the IP packet will be dealt with. For example, priority can be defined, which helps the receiver decide which packet to discard first (if it must discard some packets). This is discussed in depth below.
* **Total Length** – The last 16 bits of the first row is used to provide the total length of the datagram.
* **Identification** – This is a unique 16-bit **packet ID**. No two packets can have the same ID. However, ever **fragment** from a single packet has the same ID.
* **Flags** – There are **three bits** used as flags. The first bit is unused.
  + **Don’t Fragment** – The second bit is , for Don’t Fragment. If this flag is set to by the source, intermediary routers are not allowed to fragment the packet. If the packet is too large for a router but the flag is set, then the router gives an error and drops the packet.
  + **More Fragments** – The third bit is , for More Fragments. This is set to if there are more fragments for this packet, i.e. this is not the last fragment.
* **Fragmentation Offset** – These 13 bits are used to identify the position of a fragment of a datagram. Thus, it helps with **reassembly**.
* **Time to Live** – The first 8 bits of the third row is commonly called **TTL**. This essentially refers to how many **hops** a packet can still travel before it must be dropped.

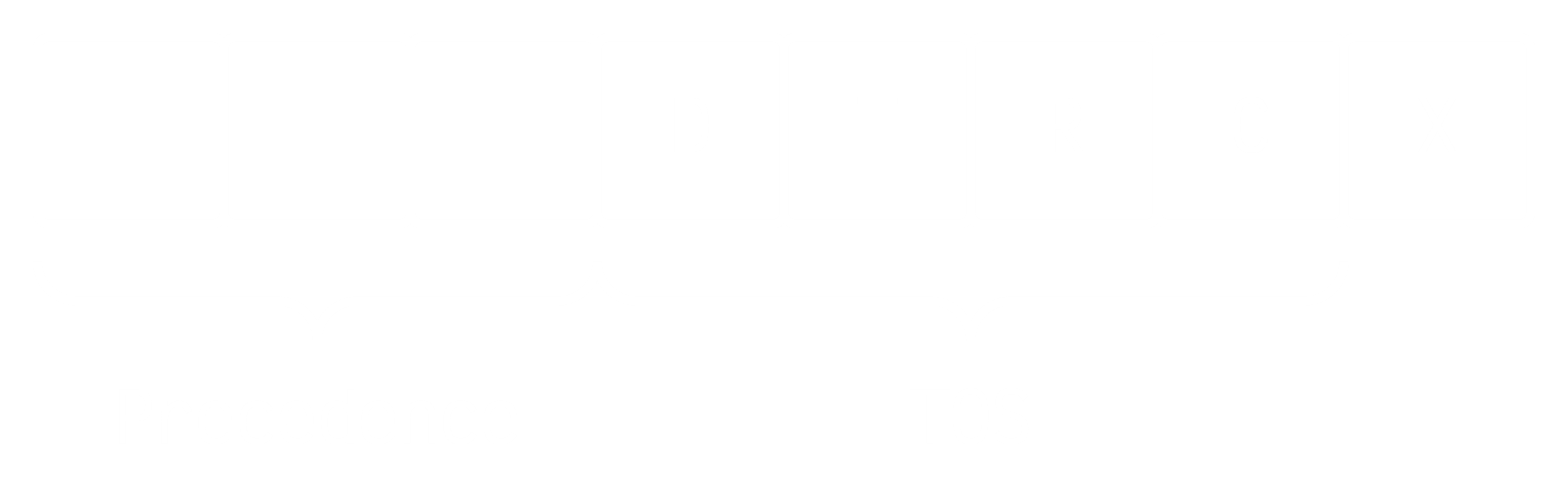
The TTL field is **mutable**, meaning it can be changed in the middle of the path. It is initially set by the source, and every router in the path **decrements** it. A packet can be delivered anywhere in the world within  **hops**, so the initial value is normally set to so that a packet can reach its destination and even come back if required. If a packet is not delivered within this limit, then there is something wrong with the packet. It may have entered a loop, in which case it will keep increasing the network load if it is not removed. Another use could be, if we set it to , then the packet is restricted to the local network.

This field used to rely on timestamps, but that required everything to be synchronized (and thus using the same clock) so it was changed.

* **Protocol** – This defines the protocol on the receiver’s end, either TCP, or UDP or any other protocols. Each protocol has a number used to identify it.
* **Checksum** – The checksum is used to detect errors in the header.
* **Source IP Address** – The source IP address takes up the 4th row.
* **Destination IP Address** – The destination IP address takes up the 5th row.
* **Options and Padding** – This is optional data used for management issues such as troubleshooting. It is discussed in depth below.

### Service Type

**Service Type** is an **8-bit** field that is used to maintain a certain **Quality of Service**, which defines how a particular packet will be treated in the network path. For example, certain packets could have a higher priority than others, and this priority would be defined in the Service Type field.



The first **3 bits** define the **precedence**, which is the priority. Thus, we can have a value between to . There could be some unforeseen circumstances under which a router is forced to drop some packets for example. In those situations, the router will decide which packets to drop based on the precedence values.

The next **4 bits** are the **Type of Service** (TOS) bits. The TOS bits actually have four values (in order)

* **Delay** (D) – The delay should be minimized.
* **Throughput** (T) – The throughput should be maximized.
* **Reliability** (R) – The reliability should be maximized.
* **Cost** (C) – The cost should be minimized.

Initially, there was a restriction that a single packet could set only one of these bits to .

The **8th bit** is **unused**.

The information above is actually an older interpretation of the Service Type field. In the new interpretation, the Service Type field is called the **Differentiated Service Type** field.

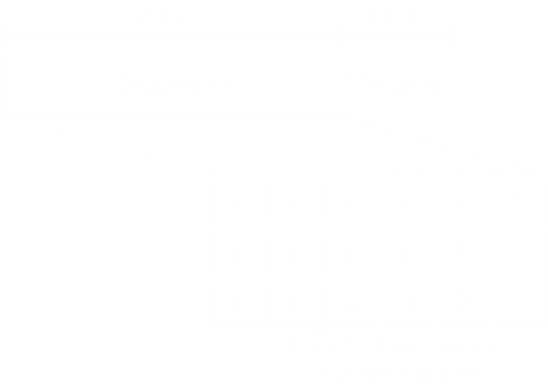


In the new interpretation, the first **6 bits** makes up the **codepoint**. There are thus possible values. Each value indicates a specific type of service.

The last **2 bits** are **unused**. Unused bits are kept in case they can be put to some use in the future. The old interpretation had only 1 bit unused, which was a bit of a problem, so the new interpretation improved upon this.



For the codepoint section, if the **rightmost 3 bits** are set to , then the **leftmost 3 bits** are interpreted as a **precedence** value. However, note that the values are interpreted using **all 6 bits**, which means the values can be , , , and so on.



For the new interpretation, the codepoints are distributed into **3 categories**:

* **Category 1** has the rightmost bit set to . This category of services is provided by a **global authority**.
* **Category 2** has the rightmost 2 bits set to . This category of services is provided by a **local authority**.
* **Category 3** has the rightmost 2 bits set to . This category of services is used for **experimental purposes**.

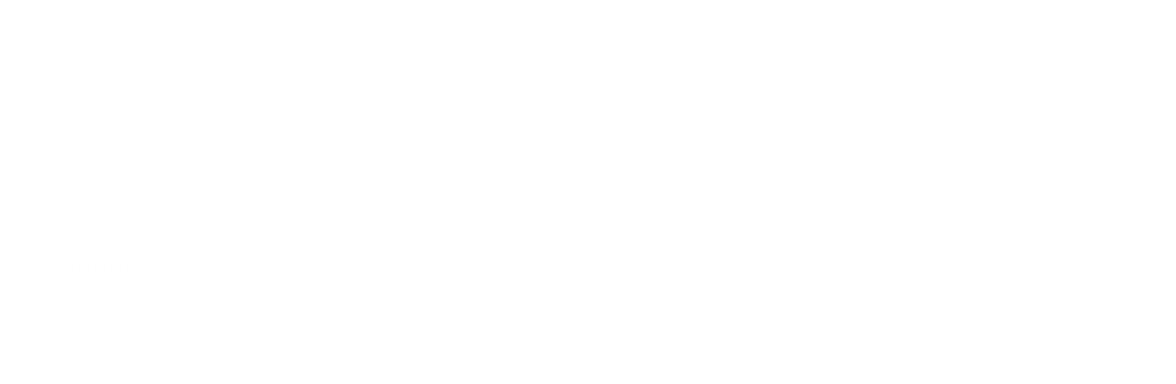
As can be seen, a complex approach is taken to dividing the range instead of dividing it into three parts. The ranges are **not contiguous**.

## 7.3 Fragmentation

For **fragmentation**, we only need to concern ourselves with the second row of the IP packet header, which contains the **Identification**, the **Flags** and the **Fragmentation Offset**.

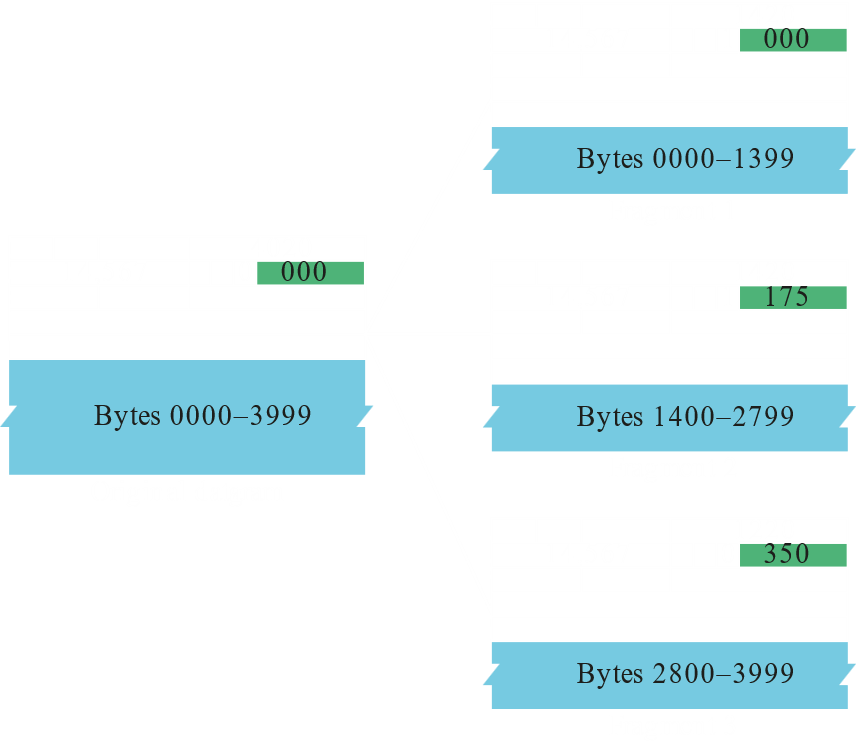
Fragmentation is necessary because the **physical networks** have limitations. They can only handle a **maximum frame size**, as defined in the **MTU table** at the source machine. The IP packet does not need to meet this limitation, but the frame does. Therefore, the IP packet needs to be fragmented. Further fragmentation can also take place at routers, but only IPv4 allows this. For IPv6, fragmentation can only take place at the source machine.

Note that only **data** in a datagram is fragmented, not the actual header.



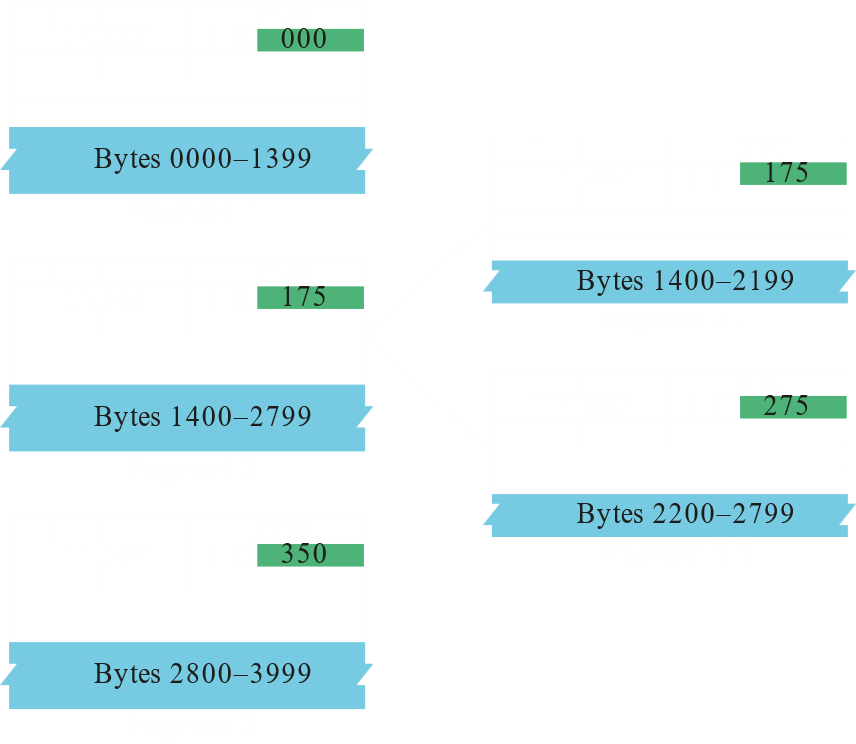
In the above diagram, we have bytes of data, which is being divided into three fragments, from to , from to and from to . For each fragment, the **offset** is given as the first byte divided by , , and respectively.

The **Fragmentation Offset** value is of **13 bits**, and we need to fit the offset value in these 13 bits. In order to do this, we are dividing by , since the original offset value cannot be fit in 13 bits. The maximum value representable by 13 bits is , and , which is why the IP datagram can be between and bytes. Thus, fragment sizes can only be multiples of .



This diagram should add some more information.

* The **total length** field changes according to the fragment size, with the actual data being bytes, bytes and bytes respectively for each fragment and the headers all being bytes.
* The **identification** number is not a concern at the moment. It is the same for all the fragments, which is how we identify that they are from the same packet.
* The **M flag** was set to in the original packet, which indicates that this is either the last fragment or that there was no fragmentation. For the fragments however, the M flag is set to for the first two fragments and for the last.

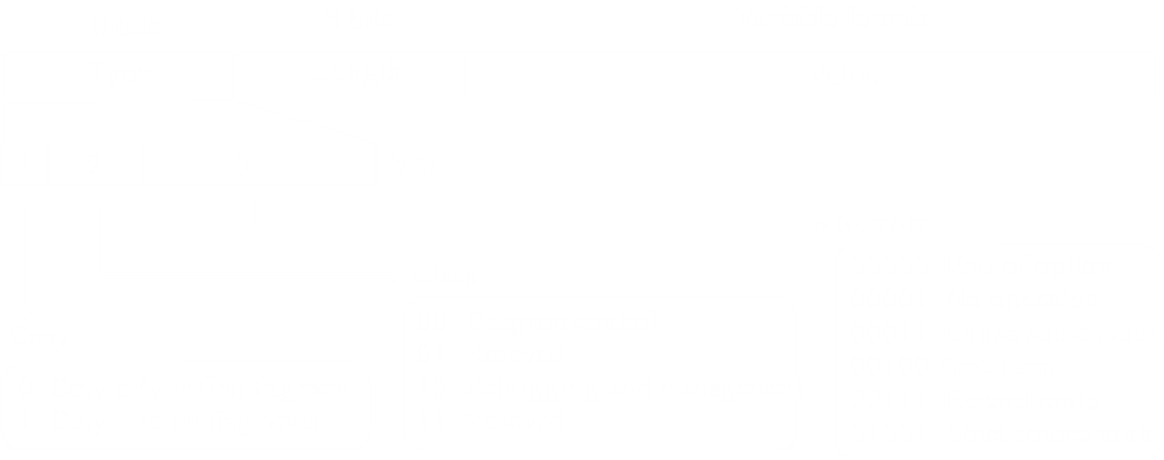


If we perform another layer of fragmentation on the second fragment (perhaps at an intermediary router), notice that the offset values correspond to the offset value for Fragment 2. Fragment 2.1 starts at , not . Since it is of 800 bytes, Fragment 2.2 starts at .

Also notice that the M flag is set to for both fragments 2.1 and 2.2, since fragment 2 had the flag set to , meaning there are more fragments after fragment 2.2.

On the receiver’s end, the **network layer** receives each fragment as though it were an individual IP packet. It then notices that the fragments have the same identification and uses the offset values to reassemble the packet. It ensures that all the fragments have arrived and there are none in between that are missing. Once this is done, it forwards the reassembled packet to the upper layers.

## 7.4 Options



The **options** field has the format shown above.

The first **8 bits** tells us about the **type**. This is divided into three parts:

1. The first bit tells us about the **copy** status. If this is set, the option will be copied to every fragment of the packet. Otherwise, it only applies to the first fragment.
2. Next, two bits define a **class**, which we will not be discussing in depth. Although 2 bits are used, only 2 of the 4 possible options are in use. The other 2 are reserved. is the **datagram control** class and is the **debugging and management** class.
3. The last **five bits** define the **number**, which tells us what the actual option is. There are 32 possible options, but we are only using **6** for now.

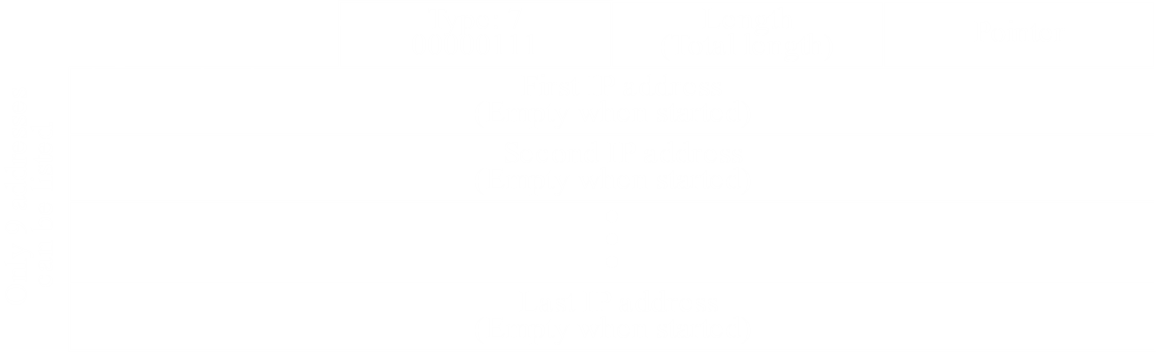
Options can be divided into two types:

1. **Single Byte** – These can be of two types:
   1. **No Operation**, which is used to align options.
   2. **End of Option**, which is used to indicate the end of the options field.
2. **Multiple Bytes** – These can be of four types:
   1. **Record Route**
   2. **Source Route**, which has two subtypes, **Strict Source Route** and **Loose Source Route**.
   3. **Timestamp**

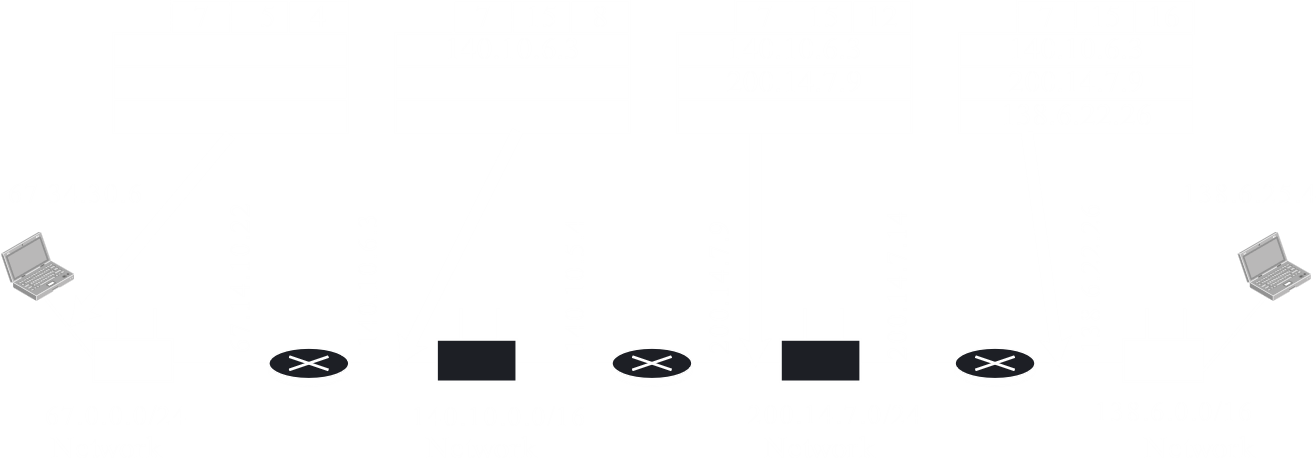
When we say that options are optional, we mean that they may or may not be included in a particular packet by the source machine. However, every machine along the path must be capable of handling all the possible options.

### Record Route

**Record route** is used **record the path** taken by an IP packet.



The maximum size of the options field can be **40 bytes**. **4 bytes** is used to give extra information, such as the **type** of the option, the **length** of the options and a **pointer**, which points to the first empty slot. The remaining **36 bytes** gives us  **IP addresses** at most. This is the maximum number of IP addresses that we can store when recording the path. In general, an IP packet travels through routers at most.



Notice that at each hop, the IP address of the hop is recorded. The **outgoing** IP address is the one being recorded. The **pointer** is also updated. It **starts at**  because the first bytes are already occupied when we start.

The value () tells us that only the **first fragment** should have the option, the option is from the **datagram control** class and that it is the **record route** option.

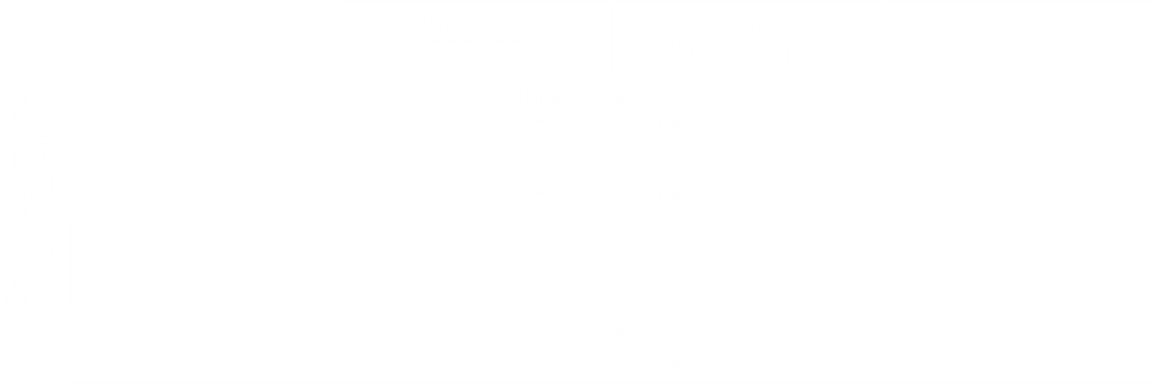
The **length**, , means that the options will take up bytes **at most**. Thus, notice that the table is full at the destination and the pointer points to , which is not accessible. If, at this point, we have still not reached our destination, something has gone wrong.

### Source Route

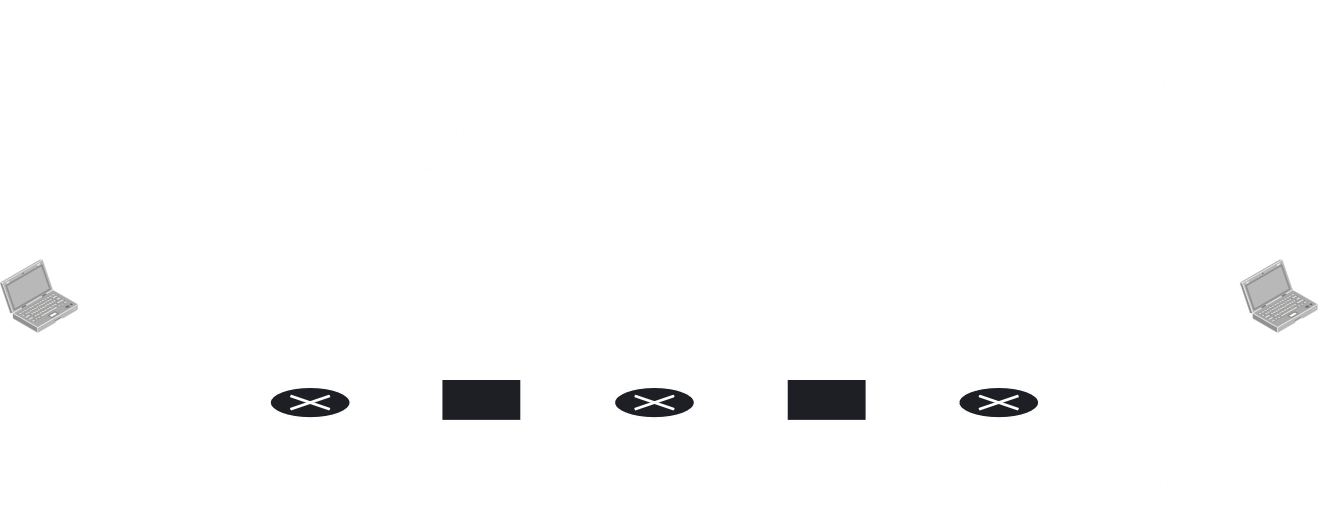
**Source Route** is used when the source machine wishes to **dictate the path** to be taken by the IP packet, as opposed to allowing the routers at each hop to decide what the next hop should be.

**Strict Source Route** specifically, forces the IP packet to go through each of the hops defined, without being able to skip any or go anywhere else.

Strict source route is somewhat similar to record route, except that the IP addresses are provided initially. Again, there can be  **at most**.



The **type** value for strict source route is (), which means the option must be copied to **every fragment**, is from the **datagram class** and is **strict source route**.



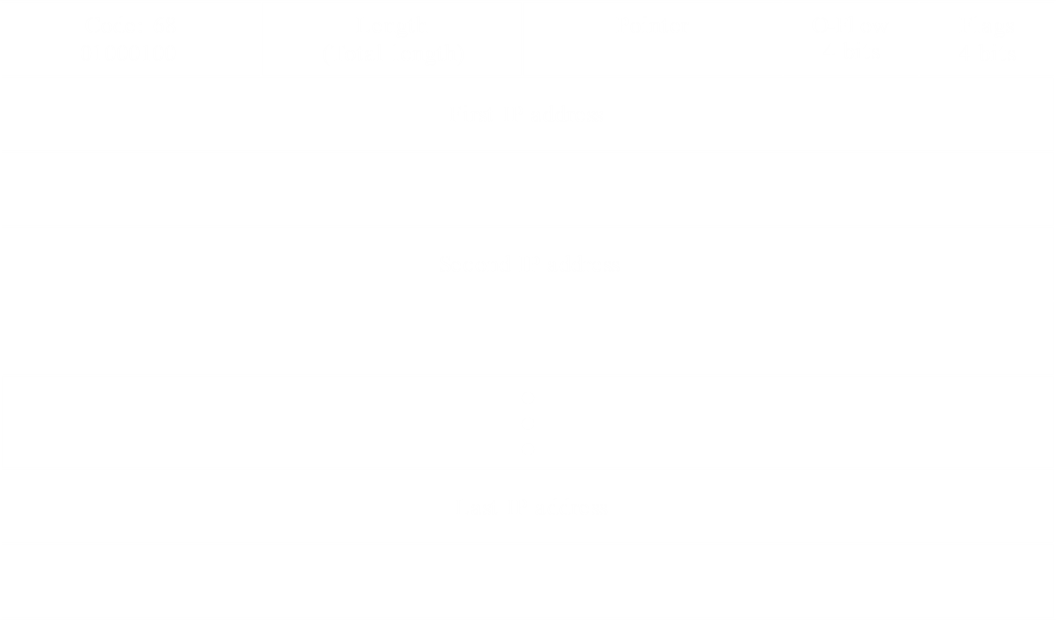
Notice that the **destination IP address** is being updated at every hop and set to the **incoming** IP address of the nest hop. The previous destination address takes the place of the next one in the table.

**Loose Source Route** forces the IP packet to go through each of the hops defined but allows additional hops as well.

Loose source route has a **type** value of () meaning it is copied to **every fragment**, is from the **datagram** class and is loose source route.

Source routing is an exceptional case which causes the destination IP address to change at each hop. Normally, the destination IP address is fixed.

### Timestamp

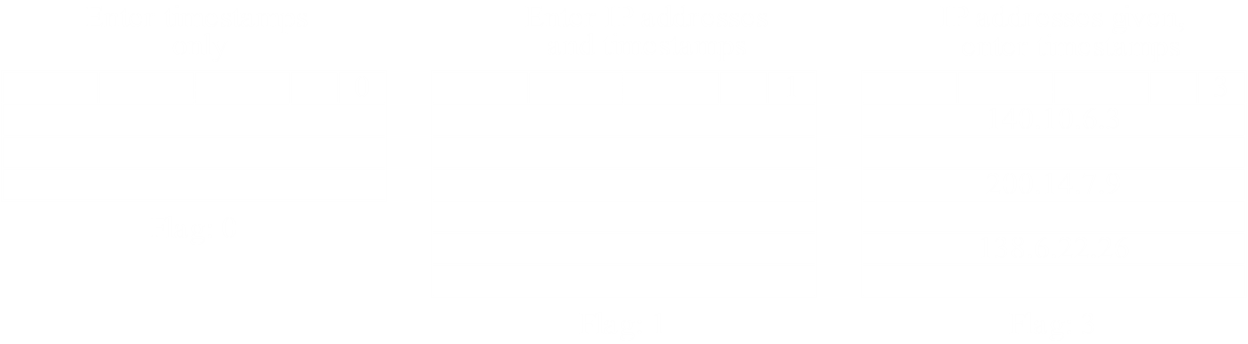


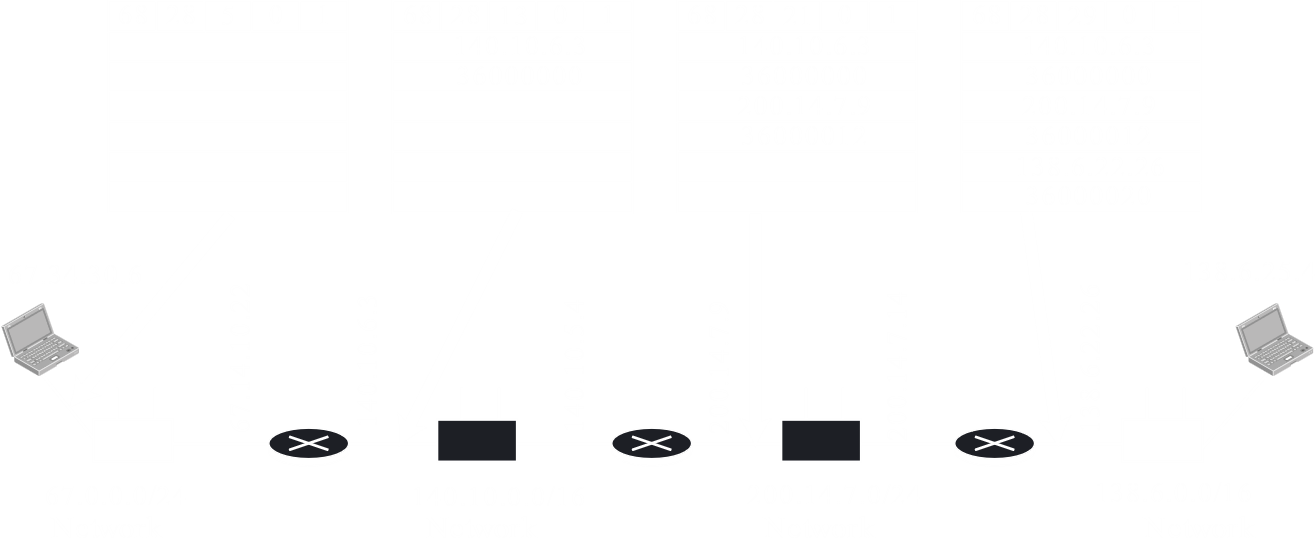
For the **timestamp** option, the **type** field has a value of (), which means it is only present in the **first fragment**, is from the **debugging and maintenance class** and is the **timestamp** option.

The setup is similar, but we now have two new fields, overflow and flags.

Whenever a packet reaches a router and the router finds that its IP address is not in the list, then it **increments** the **overflow** field.

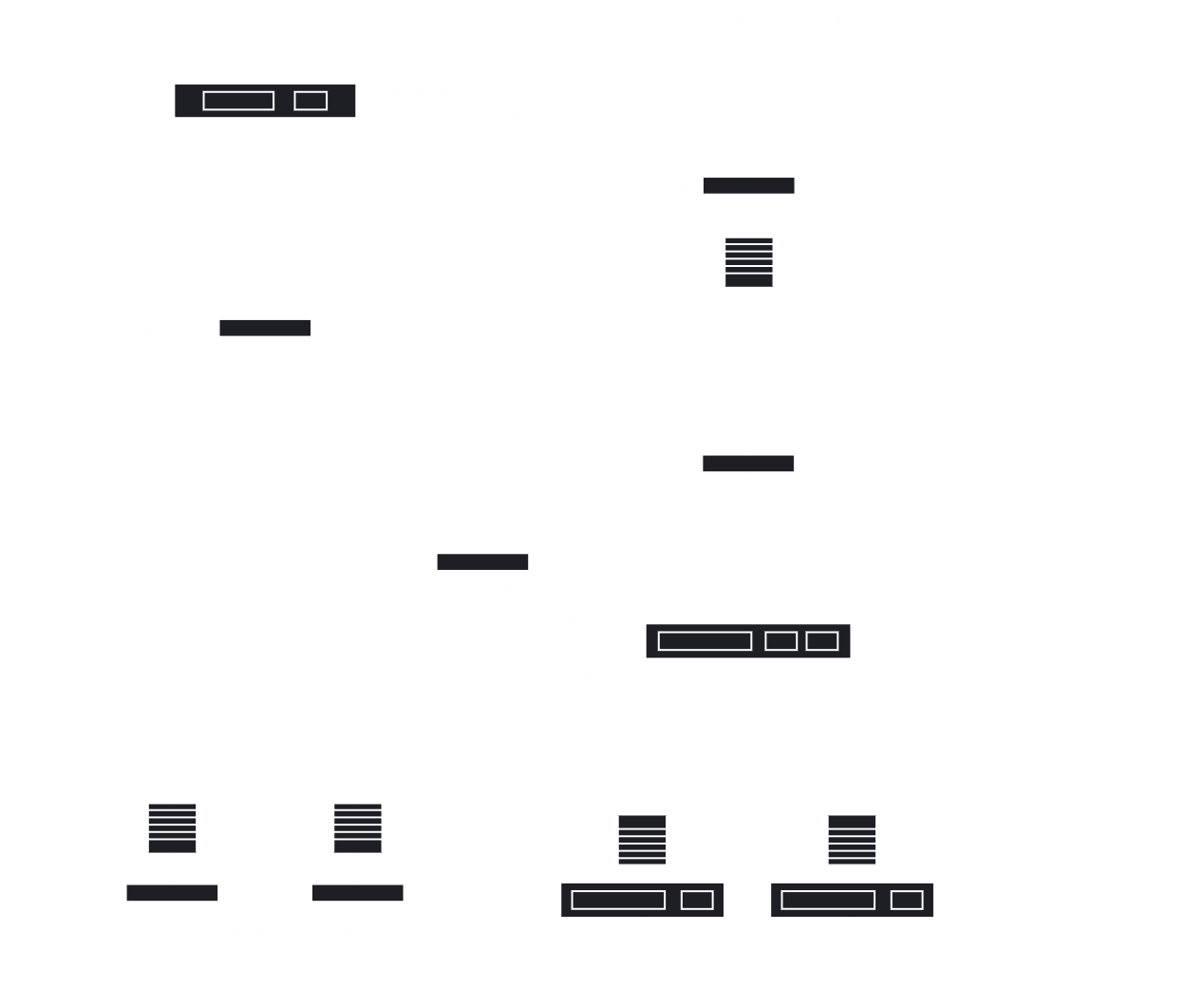
The **flag** field dictates how the IP addresses will be stored. A value of means only the timestamps are recorded, a value of means both the IP addresses and the timestamps will be recorded and a value of means the IP address are provided, to be used with strict source routing, and just the timestamps should be recorded.





## 7.8 IP Package

The different concepts we saw above can be seen in a single diagram.



Each major action is handled by a **module**.

1. The **header adding module** creates the IP packet.

IP\_Adding\_Module (data, destination\_address)  
{  
 Encapsulate data in an IP datagram  
 Calculate checksum and insert it in the checksum field  
 Send data to the corresponding queue  
 return  
}

PSEUDOCODE

1. The **processing module** processes IP packets to decide what to do with them.

IP\_Processing\_Module (datagram)  
{  
 Remove one datagram from one of the input queues  
 if (destination address matches a local address)  
 {  
 Send the datagram to the reassembly module  
 return  
 }  
 if (machine is a router) decrement TTL  
 if (TTL <= 0)  
 {  
 Discard the datagram  
 Send an ICMP error message  
 return  
 }  
 Send the datagram to the forwarding module  
 return  
}

PSEUDOCODE

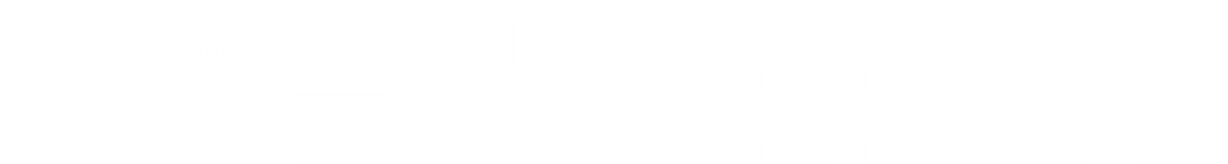
1. Packets that have arrived are sent to the **reassembly module** in case fragments need to be reassembled.

IP\_Reassembly\_Module (datagram)  
{  
 if (offset value == 0 && M == 0)  
 {  
 Send datagram to the appropriate queue  
 return  
 }  
 Search the assembly table for the entry  
 if (entry not found) create new entry  
 Insert datagram into the linked list

if (all fragments have arrived)  
 {  
 Reassemble the fragments  
 Delivery the fragments to the upper-layer protocol  
 return  
 }  
 else  
 {  
 if (time-out expired)  
 {  
 Discard all fragments  
 Send an ICMP error message  
 }  
 }  
 return  
}

PSEUDOCODE

The reassembly module has a **reassembly table** which stores the fragments in a linked list alongside some extra information.



1. Packets that are being sent out are sent to the **forwarding module**, from where it goes through the **fragmentation module** in case it needs to be fragmented.

IP\_Fragmentation\_Module (datagram)  
{  
 Extract the size of the datagram  
 if (size > MTU of the corresponding network)  
 {  
 if (D bit is set)  
 {  
 Discard datagram  
 Send an ICMP error message  
 return  
 }

else  
 {  
 Calculate maximum size  
 Divide the segment into fragments  
 Added header to each fragment  
 Add required options to each fragment  
 Send fragment  
 return  
 }  
 else Send the datagram  
 return  
 }  
}

PSEUDOCODE