Datagram

Version	H. len	Service	Datagram length	
Datagram identifier			FR-FR	FR-FR-FR-FR
Time-to-live		Transp. prot.	H. Checksum	
Source IP address				
Destination IP address				
Options				
Data				

Each line represents a 32–bit word; the second word deals with fragmentation/reassembly.

IP datagram fields

Version: IPv4 = 4 (IPv6 = 6).

Header length: (in words) variable because of options. Typically 20 bytes (i.e. 5).

Type of Service: used by some routers as dropping priority.

Datagram length: in bytes. Most implementations of IPv4 limit datagrams to 576 bytes.

Identifier: a 16–bit tag associated with the datagram. Typically used for fragmentation/reassembly.

Time–to–live: expressed in hops. This field is decremented by one after each hop; the datagram is dropped when TTL becomes 0.

Protocol: TCP=6, UDP=17, ICMP=1, etc. (list in IANA).

Options: not used much, but potentially useful.

Type of service

This a 4–bit mask. The meaning of the bits is:

0001	minimise cost
0010	maximise reliability
0100	maximise throughput
1000	minimise delay

TOS can be any combination of these four options; the interpretation of any combination is undefined.

There are 3 more flags for datagram priority; they have never been implemented.

Fragmentation and reassembly

	Datagram length	
Datagram identifier	Flags	Fragmentation offset

Identifier: a 16–bit tag associated with the original datagram. Each fragment will carry the same tag.

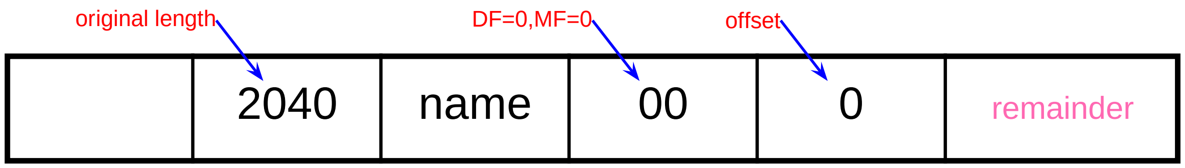
Flags: two relevant bits: DF=“do not fragment” and MF=“more fragments follow”

Fragmentation offset: a 13–bit field that gives the offset of the beginning of this fragment. It is measured in 8–byte blocks (not bytes!).

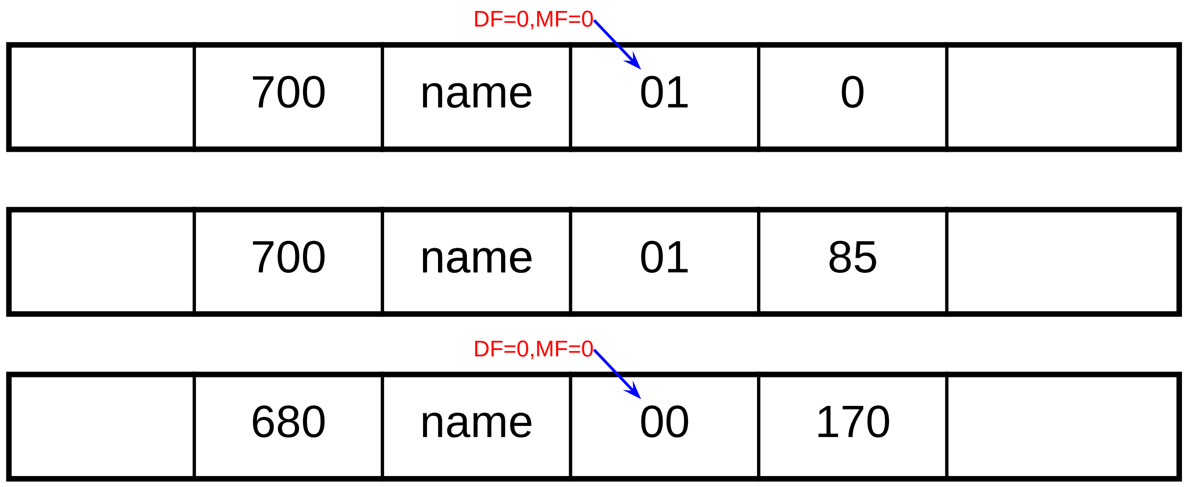
TCP submits a segment with a payload of 2000 (the segment's length is 2020).

This instance of **IP** limits a datagram size to 700 bytes.

Without fragmentation, the datagram would look like this:



With fragmentation, there will be 3 datagrams of total length $2020 + 3 \times 20$.



These datagrams will be reassembled at the destination with the help of the (supposedly) unique identifier.

Routing table

IP decides where to route a datagram based on its routing table in which the most up–to–date information about the Internet is kept. **IP** is only a passive user of this table–it is updated by auxiliaries or manually.

The format, the contents, and the scope of the r.t. entries is not standardised (it does not need to be, as the table is not accessed by any outside software, only by **IP** and its auxiliaries.

Most tables look very similar: they have 4 to 8 columns with the basic 4 always present:

Destination: ((“network”) contains the network address that this entry describes—all the packets addressed to this network will be sent this way unless there are several matching entries (CIDR prefixes).

Network mask: to be applied to the address looked for before matching with the 11destination” field.

Gateway: (“next hop”) is the **IP** address of the router to which this packet should be forwarded to.

Interface: the link interface for the next hop.

Possible additional fields:

Flags: indicate the current state of the destination:

- **U** means the link is **up** (the other end of the link is up and running). If not present, this table entry must be ignored.
- **G** the destination is in another network and must be sent to a **gateway**. If not present, the interface listed is part of the local subnet.
- **H** the “destination” field is the address of a host, not a network address.
- **D** and **M** indicate that this entry was touched by **ICMP**.

Metric: the **cost** of this link, usually expressed in number of hops to destination. By convention a metric of 16 or more means that the route is unusable.

Use: the number of datagrams that were forwarded using this entry.

A routing table as displayed by netstat -rn (Linux).

```
netstat -rn
Kernel IP routing table
Destination    Gateway        Genmask        Flags Metric Ref Use Iface
131.104.48.0   0.0.0.0        255.255.254.0  U     0      0     0 eth0
127.0.0.0      0.0.0.0        255.0.0.0      U     0      0     0 lo
0.0.0.0        131.104.48.18  0.0.0.0        UG    0      0     0 eth0
```

The entry “`lo`” stands for “loopback” (or for “local”).

Default route

By convention, one entry in the routing table acts as default route. It has a destination address of $0.0.0.0$ and a mask of $0.0.0.0$ which makes it match every possible **IP** address. The default route points to a gateway to the rest of the Internet; all the datagrams not claimed by the other routing table entries will be forwarded this way.

Address depletion

It is increasingly common that an organisation received a block of **IP** addresses from IANA (ICANN) some years ago. The organisation grew since then manifold—to the point that this block cannot meet the needs of the organisation.

This is particularly common in countries that were slow in deploying Internet subnets.

There are 3 ways to alleviate this problem:

- Get a bigger block of addresses and move (rename) all the hosts to this new block. Two problems: big blocks are hard to get nowadays **and** address relocation means loss of business.
- Get a second block and use both the old and the new. With clever aliasing it can be almost transparent to application–layer users, but numerous problems will emerge because the blocks are considered to be two distinct networks by the Internet (consider firewalls or hot–potato routing).
- Build a large private network with its own **IP** address space. This network will use the existing block of addresses to connect to the Internet.

Network Address Translation

NAT allows an organisation to have a large set of addresses internally and a small set of addresses externally.

Internal traffic will use the internal–large–set of addresses; these addresses do not have to be allocated by IANA because they are not visible from the Internet, hence the internal set can be as big as desired.

Handled with care, any addresses can be “hijacked” for internal use, but not without serious translation issues.

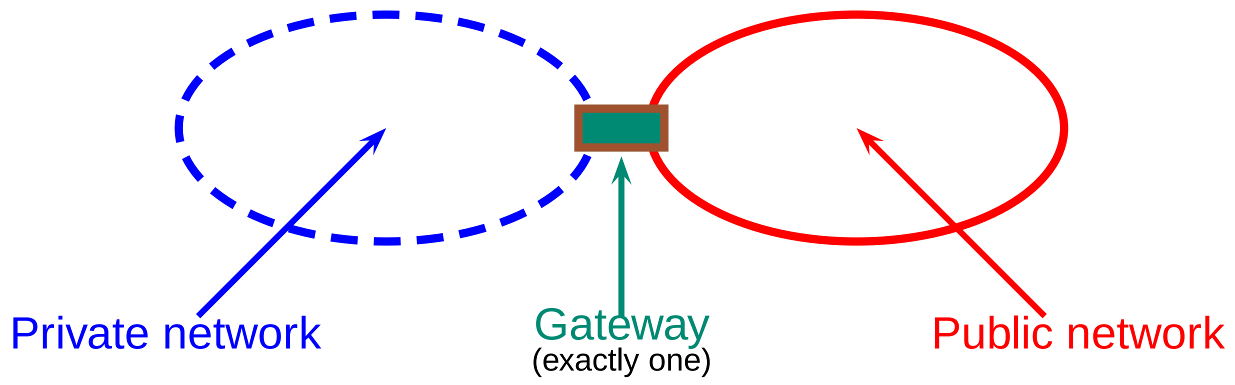
To help, IANA set aside 3 “private networks” blocks of addresses; these addresses are not legal for any host connected directly to the Internet:

Range	Size
10.0.0.0/8	2^{24}
172.16.0.0/12	2^{20}
192.168.0.0/16	2^{16}

Internet routers do not recognise these “private” addresses as legal and drop packets which carry these addresses in their IP headers.

Architecture of a private network

In order to function, a private network (using **NAT**) must be isolated from the Internet:



with a single gateway connecting the private network to the "public" Internet. This gateway must, among other duties, convert private addresses into some legitimate "public" addresses.

Note that this gateway has at least 2 IP addresses, one in the private network (say, 10.12.34.1) and another in the public Internet (say 56.67.78.89).

Internal traffic

Packets with a local destination (inside the private network) are handled normally–the organisation’s internal routers will accept the destination addresses (which are private) as long as they do not leave the private network,

Communication with the outside

The problem is with traffic going out into the Internet or, even more so, with traffic coming in from the Internet.

Traffic going out is easy to handle if the organisation uses one of the IANA private blocks. Otherwise it is handled in the same way as traffic coming in.

Traffic going out

Suppose that a local host with a private IP address of 10.12.34.56:1001 sends a TCP segment to a public address 131.104.49.122:4950.

The gateway (which will choose to use the public address 56.67.78.89) will change the source address of the packet to 56.67.78.89:x; both the TCP and the IP headers will be changed. The modified packet is sent out while the gateway records in a table the relationship:

$$10.12.34.56{:}1001 \longleftrightarrow x \longleftrightarrow 131.104.49.122{:}4950$$

Incoming traffic

Consider that the entity at 131.104.122:4950 replied to the packet just sent. The reply will carry as source address 131.104.122:4950 (obvious) and as destination address 56.67.78.89:x (because that is what it received).

When this packet reaches 56.67.78.89, the gateway will look into its table and deduce that the true destination of the packet is 10.12.34.56:1001. The headers will be changed and the packet sent into the private network.

Incoming traffic

A datagram crossing the Internet boundary will also contain another incorrect field that will have to be changed:

- TCP and UDP segments carry a checksum that includes an “imaginary IP datagram header” and these segments may be fragmented by IP.
- When a segment comes to the boundary of the private network, its destination address is changed and the checksum (over the whole segment) must be recomputed. This can be done only if one router sees the whole segment.

Hence, all the external traffic must come into the private network through **one** gateway. This gateway may use many **IP** addresses (the whole block allocated by IANA).

Translation of the destination address

The gateway keeps track of all the connections established between internal and external hosts. This solves the translation problem because the connection is uniquely identified by a port number. Note that the gateway may renumber the port number of the internal end of the connection for uniqueness.

If the datagram is connectionless, the situation is hopeless. NAT imposes the requirement that connectionless exchanges must be initiated from within the private network and expects that connectionless datagrams coming from the outside are responses to these requests. Errors are unavoidable.

Port number manipulation

Consider a private network $131.104.48.0/20$ with a gateway seen from the outside as $131.104.48.1$ and from inside as $192.168.0.1$.

Suppose that two internal hosts, $192.168.1.101$ and $192.168.2.101$ connect (in this order) to the same external server $212.58.226.33 : 80$. By accident both clients chose port 10000.

The gateway will forward the first connection request as originating from $131.104.48.1$: x and the second as originating from $131.104.48.1$: y where x and y are different.

The entries in a translation table will help direct the flow of segments back to the internal ends of the two connections.

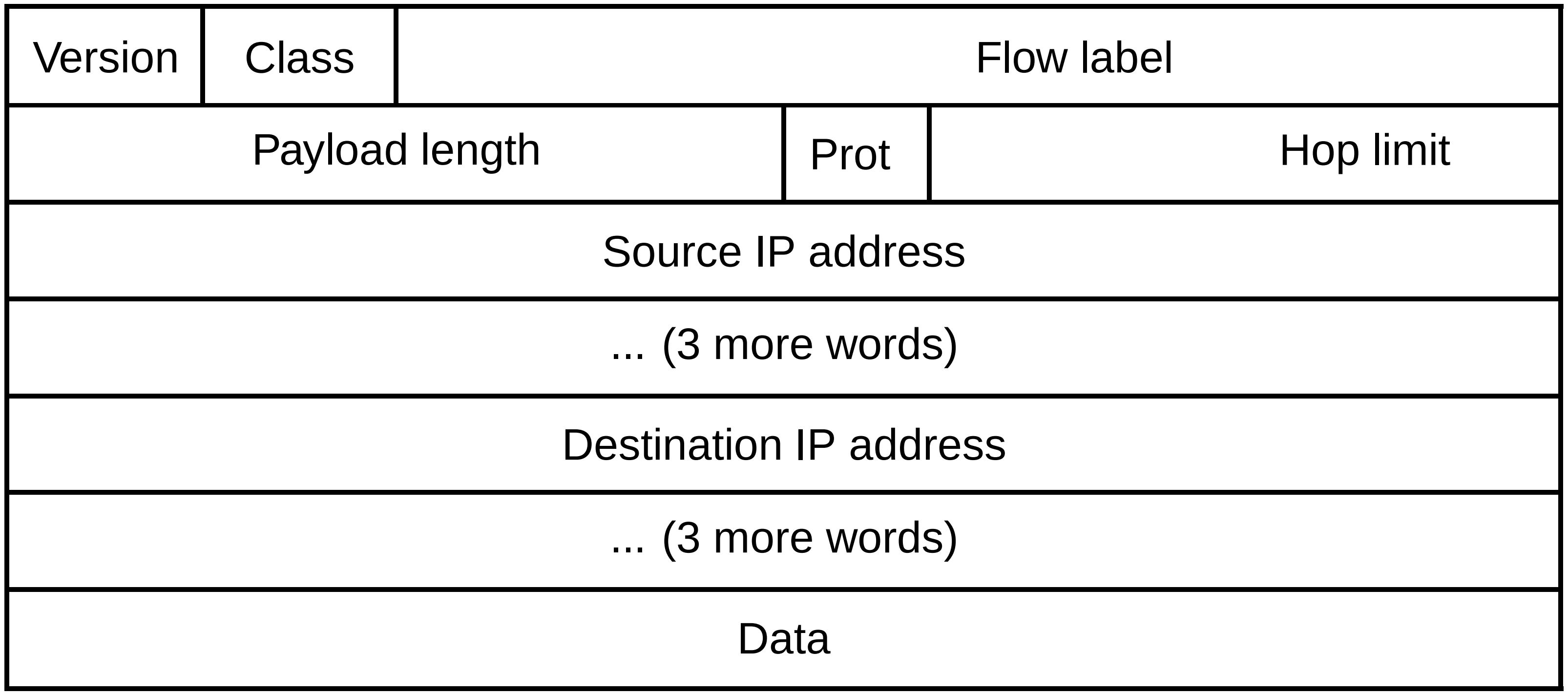
$192.168.1.101 : 1000$	x	$212.58.226.33 : 80$
$192.168.2.102 : 1000$	y	$212.58.226.33 : 80$

The same works for two connections originating from two ports of the same internal host.

IPv6

- 128–bit addresses.
- Pseudo–connections (*flows*). Details not defined.
- Fixed–size 40 byte header.
- No fragmentation/reassembly, no options, no header checksum.

Datagram


Version
Class
Flow label
Payload length
Prot
Hop limit
Source IP address
... (3 more words)
Destination IP address
... (3 more words)
Data

Compatibility of v4 and v6

Some people make a big deal out of the lack of direct compatibility of v4 and v6. Solutions:

Dual stack: the first 4 bits of the datagram give the version number. Two independent network layer protocols coexist, each handling its version.

Tunnelling: If a v6 datagram has to pass through a network area that has only v4 service, it can be encapsulated in a v4 datagram which is sent to the first v6 router past the cluster of v4 routers. This router recognises the encapsulation, extracts the v6 datagram, and forwards it.

Multiple headers

IPv6 allows multiple headers: the base header (i.e. first) has a field called *Next header* (“Protocol” in the figure). This filed is set to TCP or small UDP if there are no other headers; otherwise, it is set to the type of *extension header* that follows. This can be repeated several times; the last header must indicate a transport layer protocol.

One of the extension header types is *Options*, which is a variable–length header.

Fragmentation

Even though IPv6 does not support fragmentation officially, it really does. See Comer's book for details.